Einführung in Visual Computing

186.822

Visible Surface Detection

Werner Purgathofer

Visibility in the Rendering Pipeline

object capture/creation

scene objects in object space

modeling

viewing

projection

vertex stage („vertex shader“)

transformed vertices in clip space

clipping + homogenization

scene in normalized device coordinates

viewport transformation

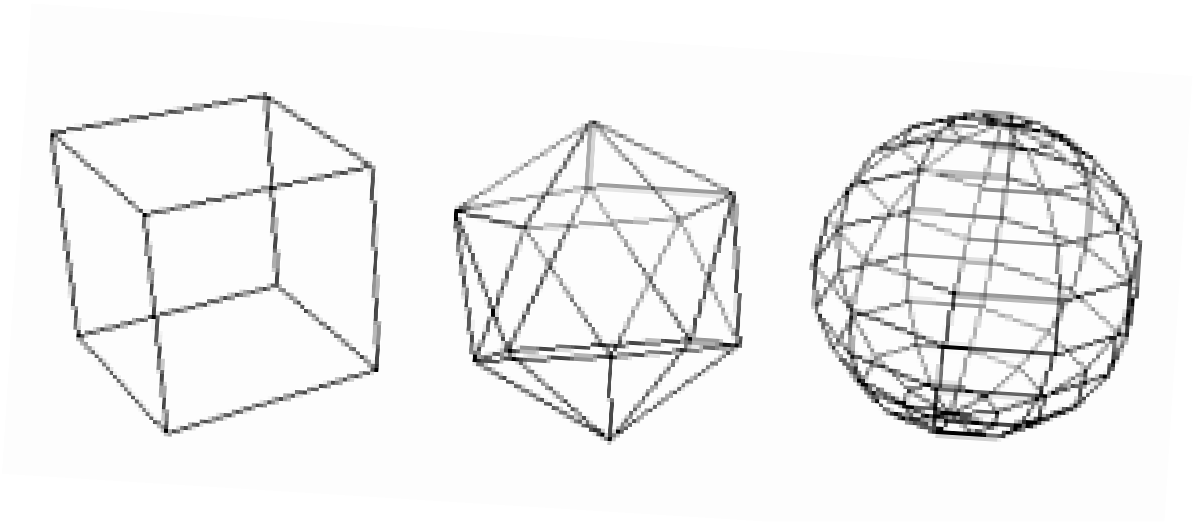
rasterization

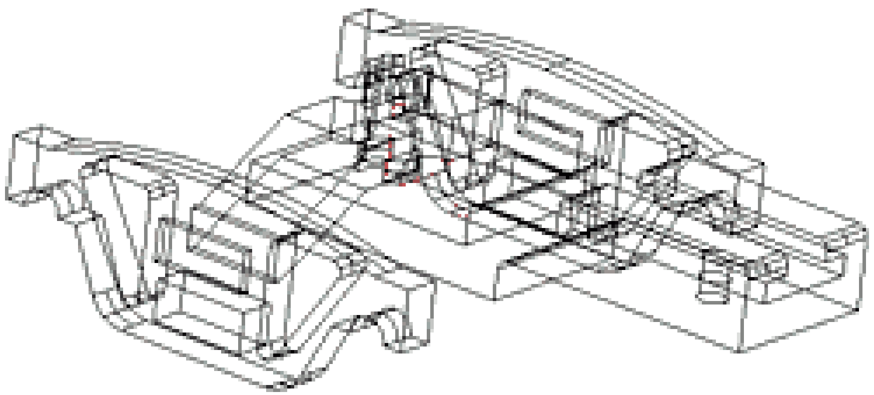
shading

pixel stage („fragment shader“)

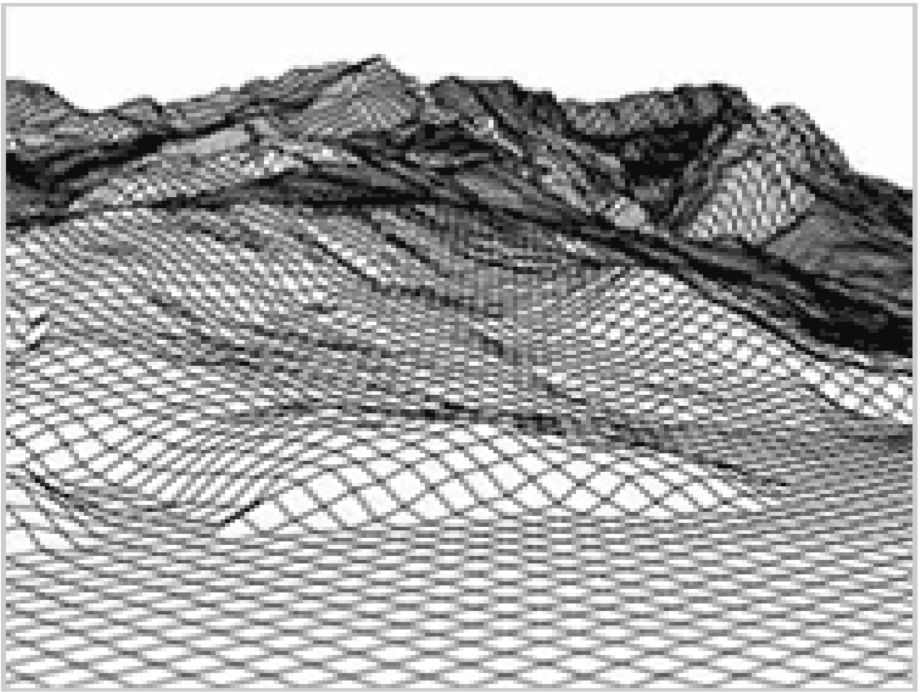
raster image in pixel coordinates

3D Display: Wireframe Display

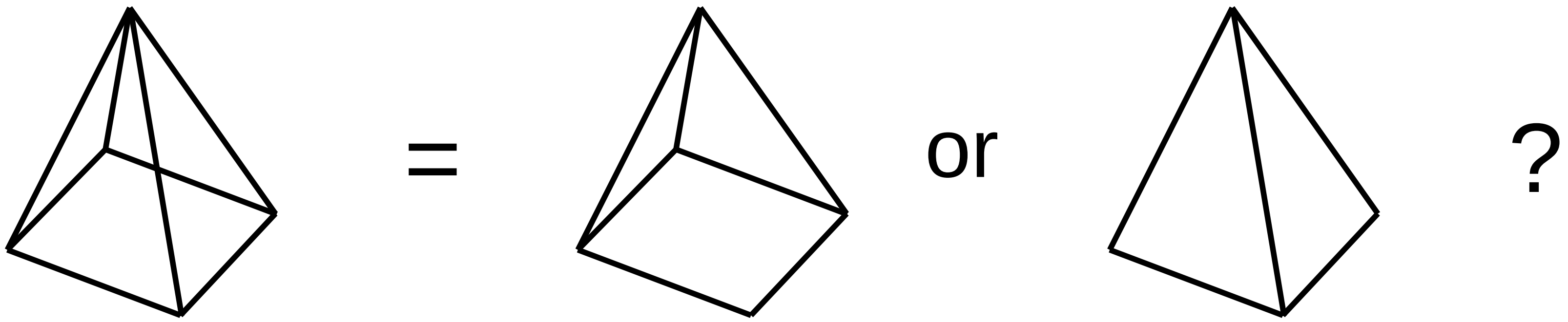




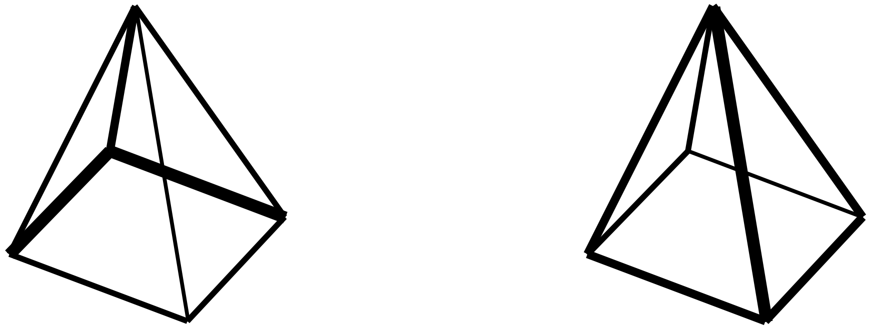
© ZwCAD Software Co.,Ltd



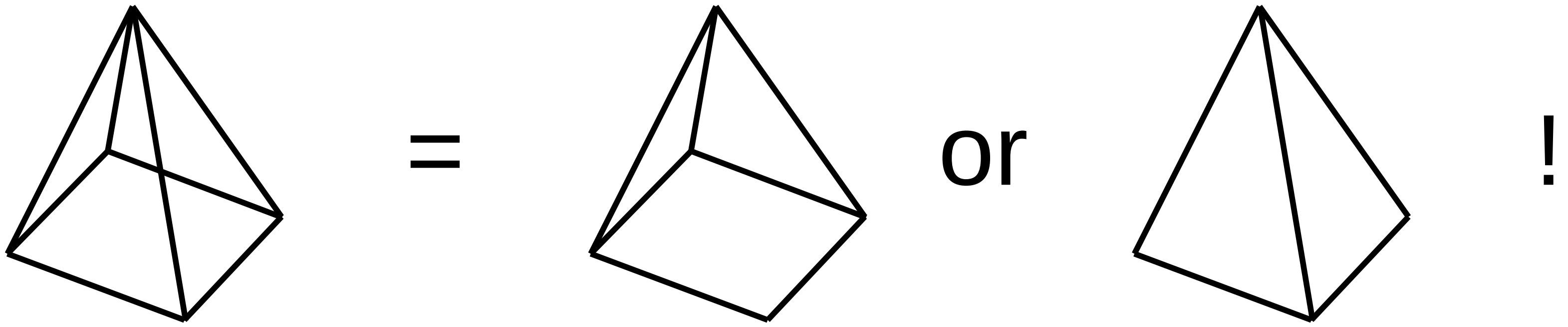
3D Display: Depth Cueing

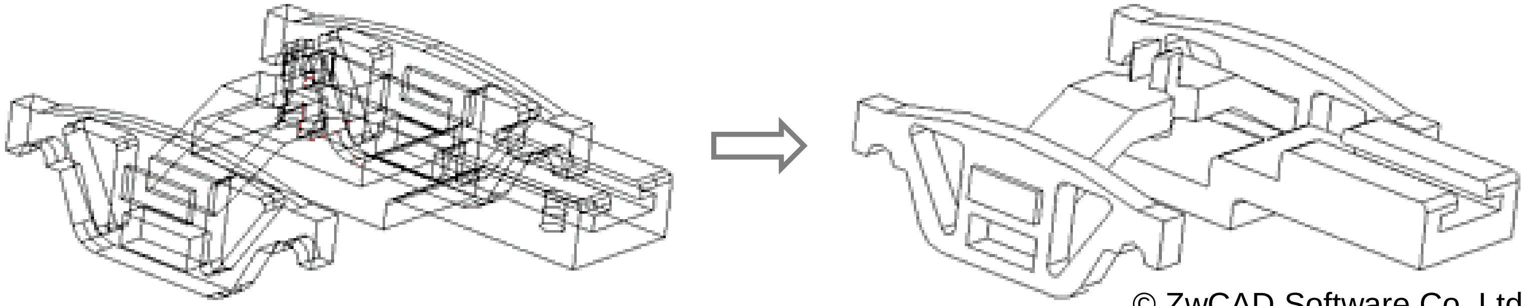


depth cueing = intensity decreases with increasing distance

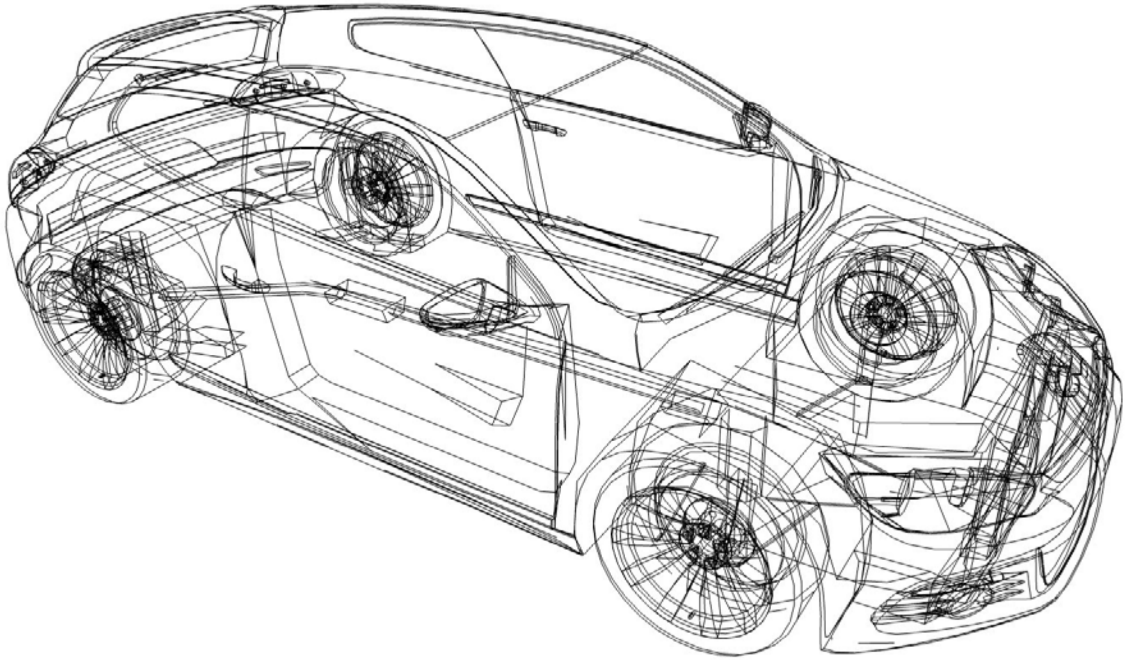


visible line and surface identification

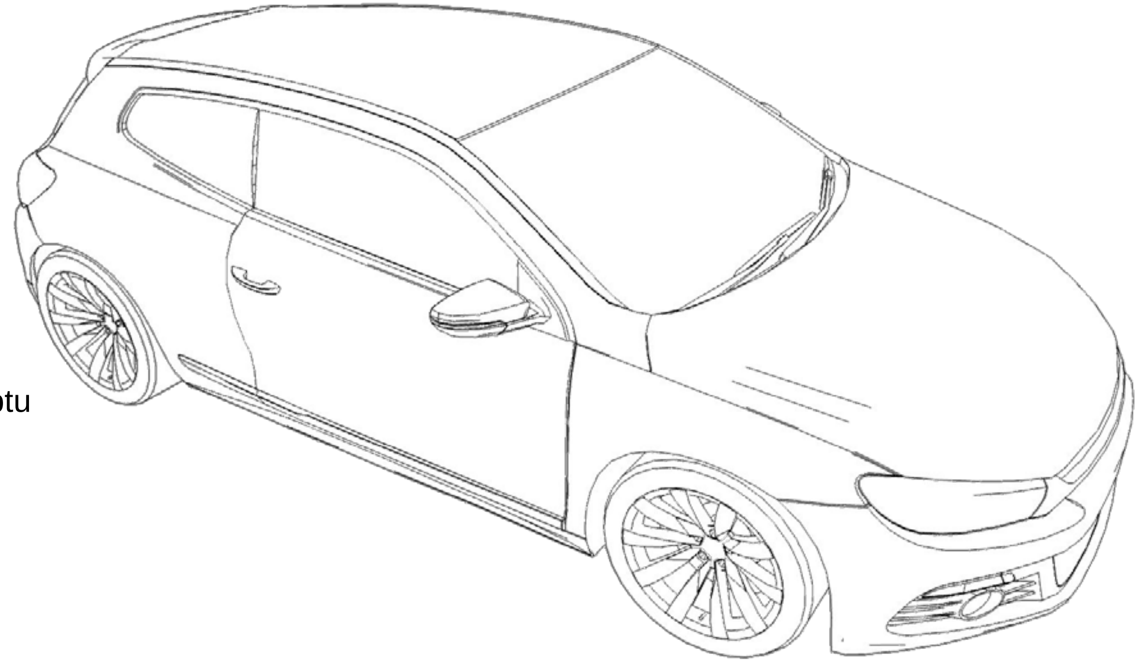




3D Display: Visibility

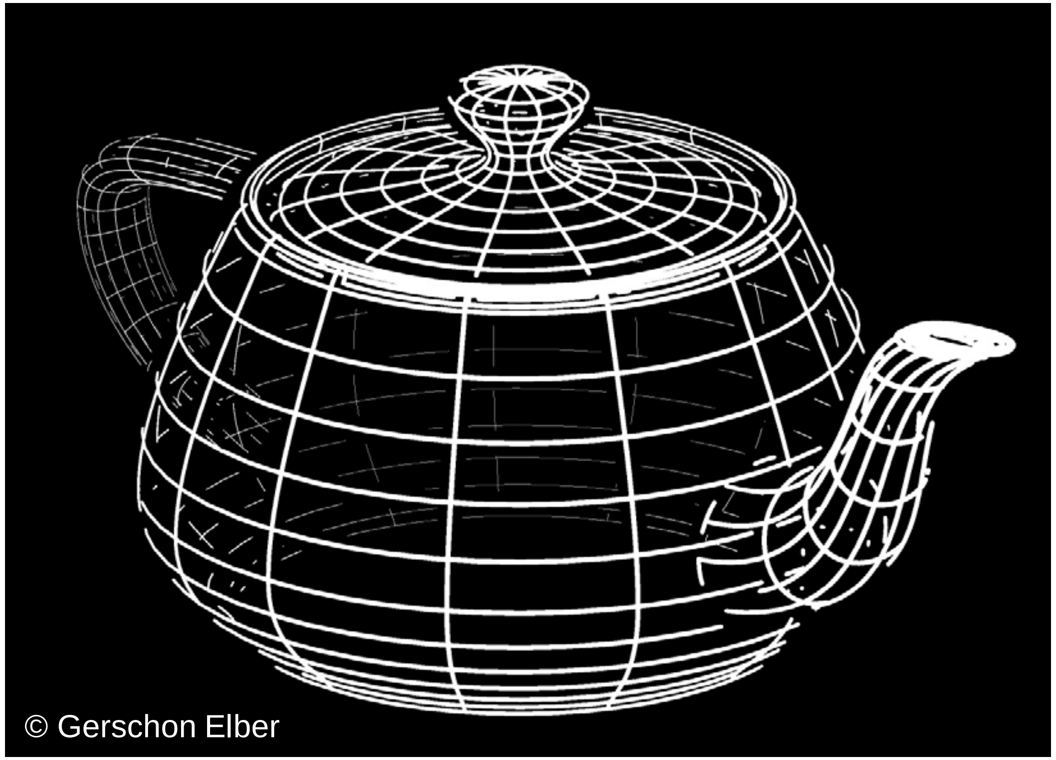


© Snaptu



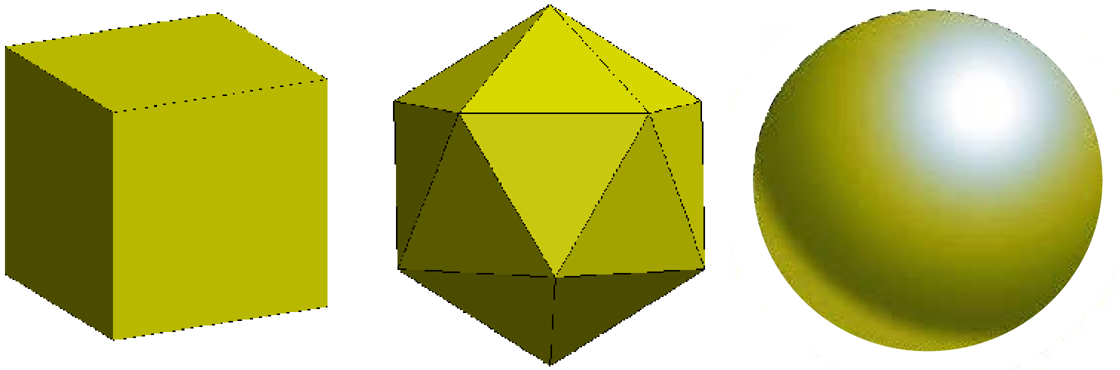
3D Display: Depth Cueing + Visibility

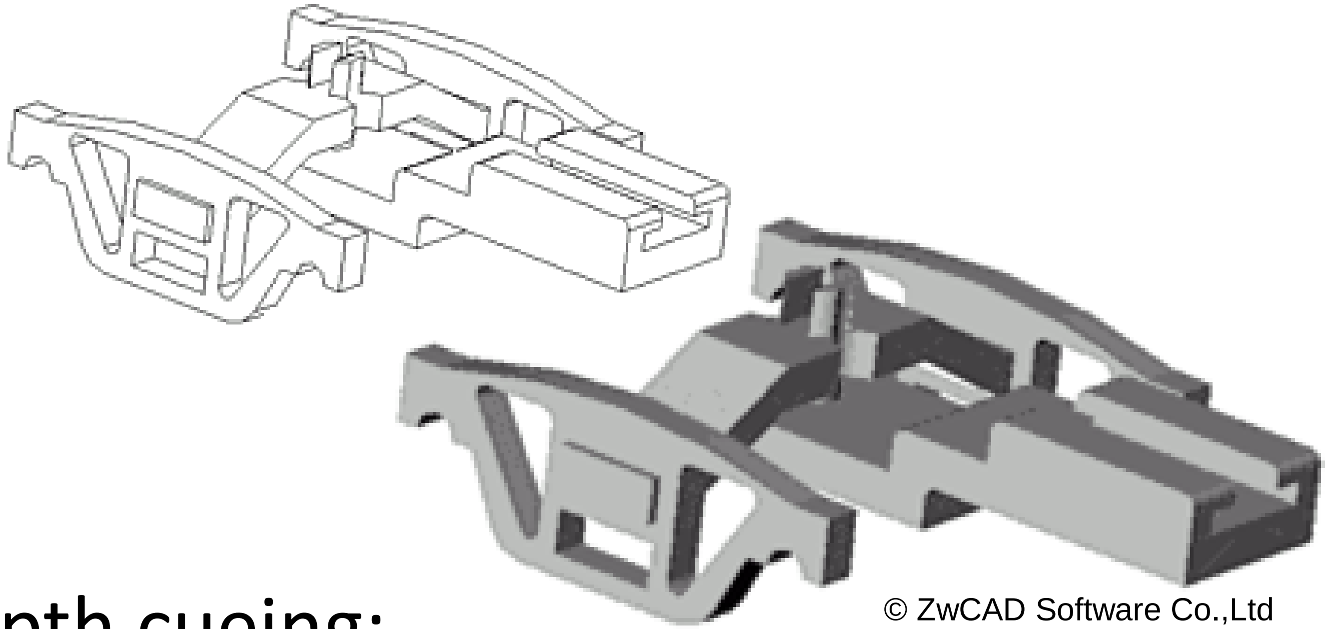
- only visible lines
- intensity decreases with increasing distance



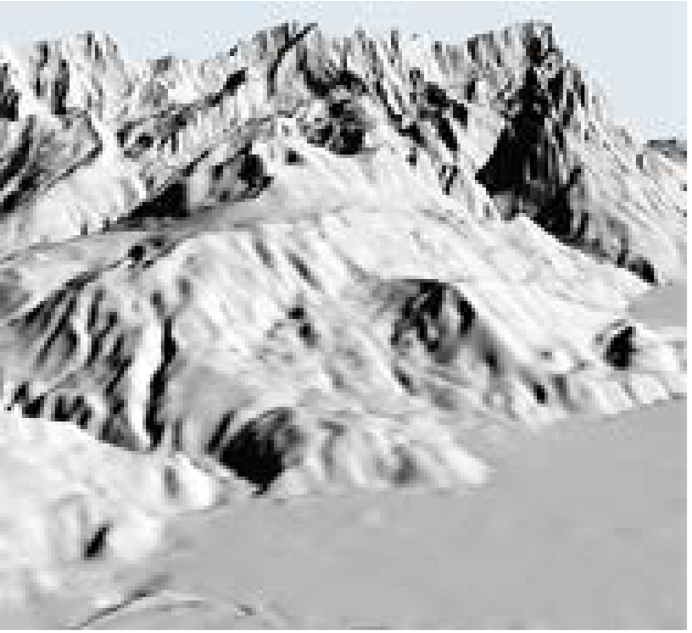
© Gerschon Elber

3D Display: Shaded Display

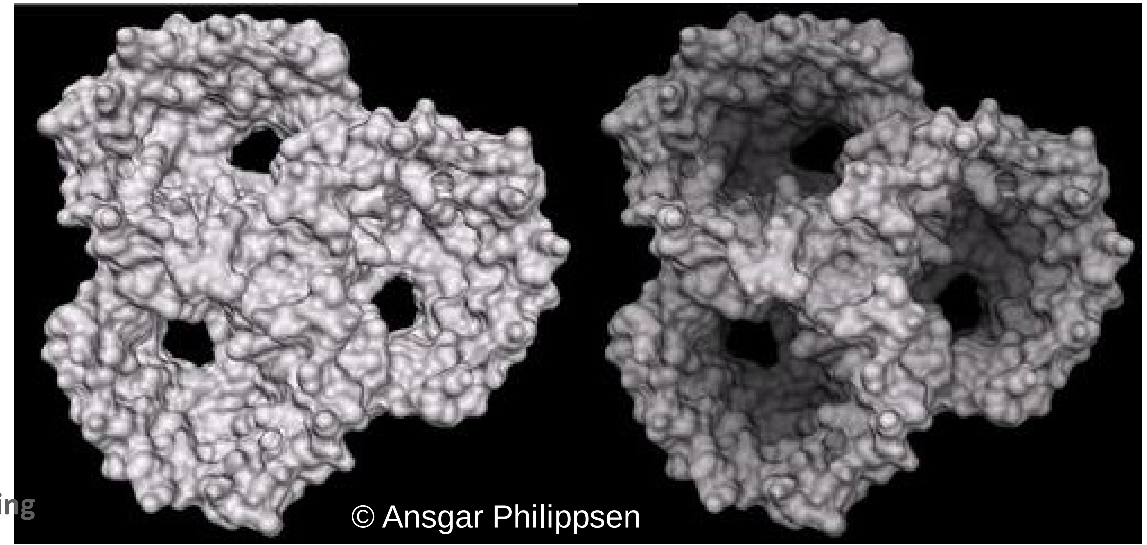




© ZwCAD Software Co.,Ltd



shading + depth cueing:



© Ansgar Philippsen

Visible-Surface Detection

identifying visible parts of a scene
(also *hidden-surface elimination*)

type of algorithm depends on:

- complexity of scene
- type of objects
- available equipment
- static or animated displays

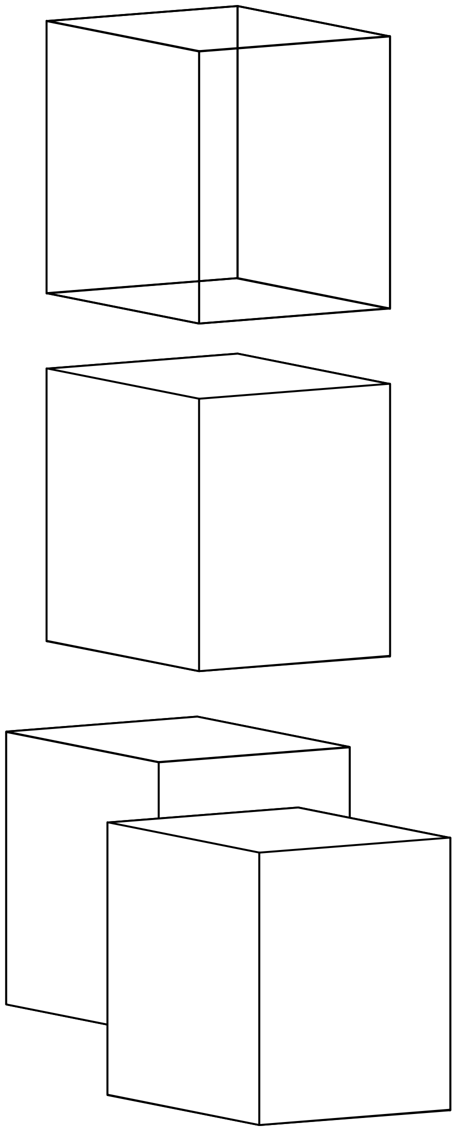
object-space methods

- objects compared to each other

image space methods

- point by point at each pixel location

often sorting and coherence used



Visible-Surface Detection Methods

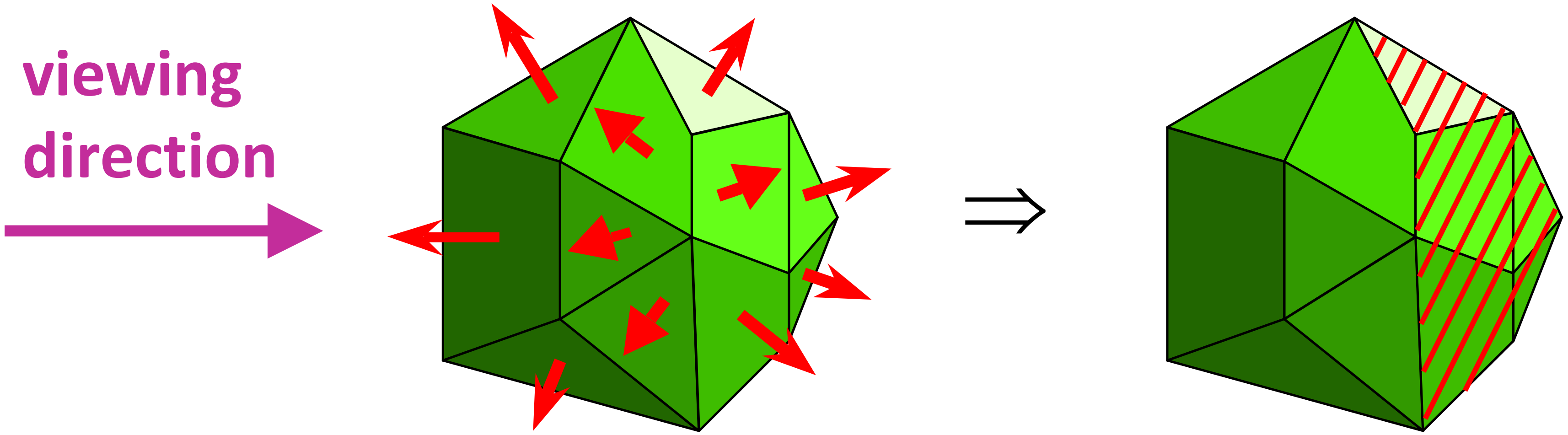
the following algorithms are examples for different classes of methods

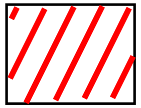
- back-face detection
- depth buffer method
- scan-line method
- depth-sorting method
- area-subdivision method
- octree methods
- ray-casting method

Back-Face Detection (1)

surfaces (polygons) with a surface normal pointing away from the eye cannot be visible (back faces)

$\Rightarrow$ **eliminate them before visibility algorithm !**



can be eliminated: 

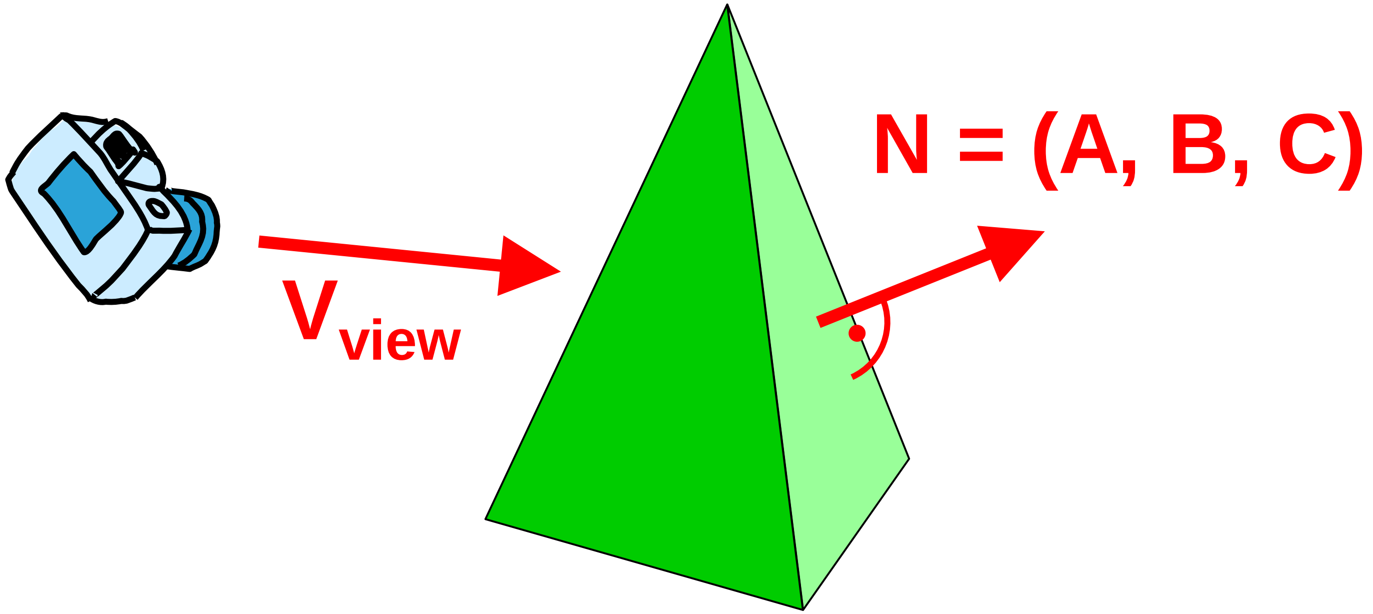
Back-Face Detection (2)

- eliminating back faces of closed polyhedra
- view point (x,y,z) “inside” a polygon surface if

$$Ax + By + Cz + D < 0$$

- or polygon with normal N=(A, B, C) is a back face if

$$V_{\mathbf{view}} \cdot N > 0$$



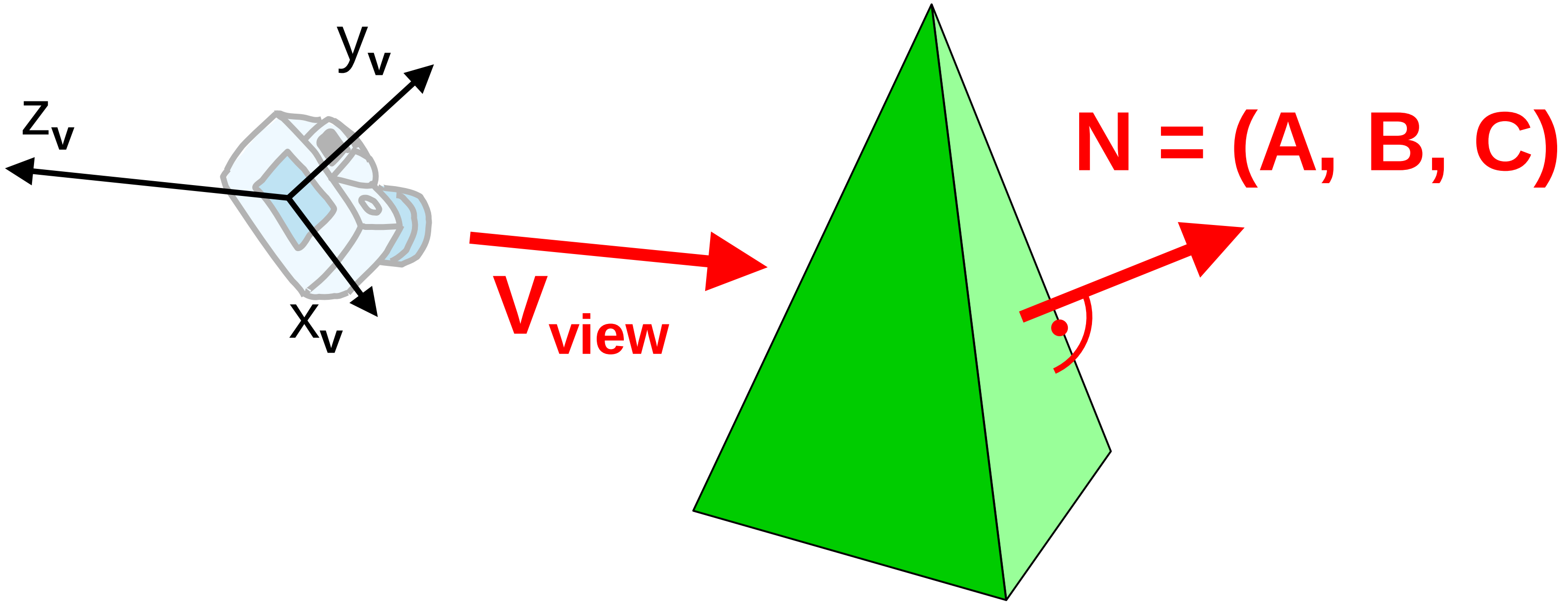
Back-Face Detection (3)

object description in viewing coordinates $\Rightarrow$ $V_{\mathbf{view}} = (0,0,V_z)$

$$V_{\mathbf{view}} \cdot N = V_z C$$

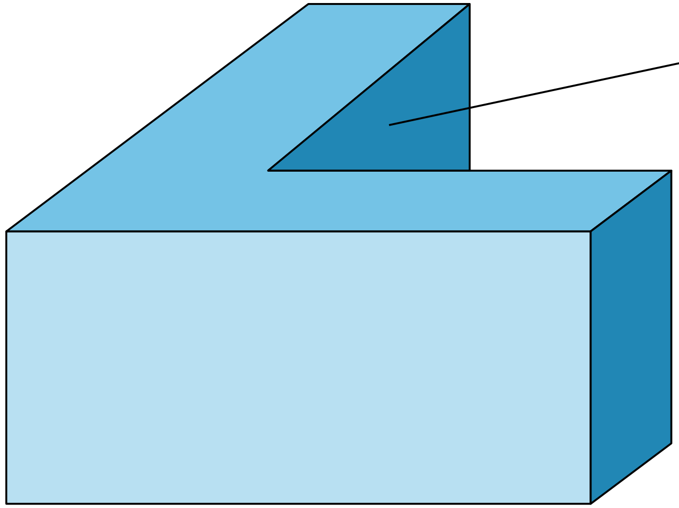
negative !

sufficient condition: if $C \leq 0$ then back-face



Back-Face Detection (4)

complete visibility test for non-overlapping convex polyhedra



face partially hidden
by other faces

preprocessing step for other objects:

about 50% of surfaces are eliminated

Depth-Buffer Method (1)

- z-buffer method
- image-space method
- hardware implementation
- no sorting!

Depth-Buffer Method (2)

two buffers:

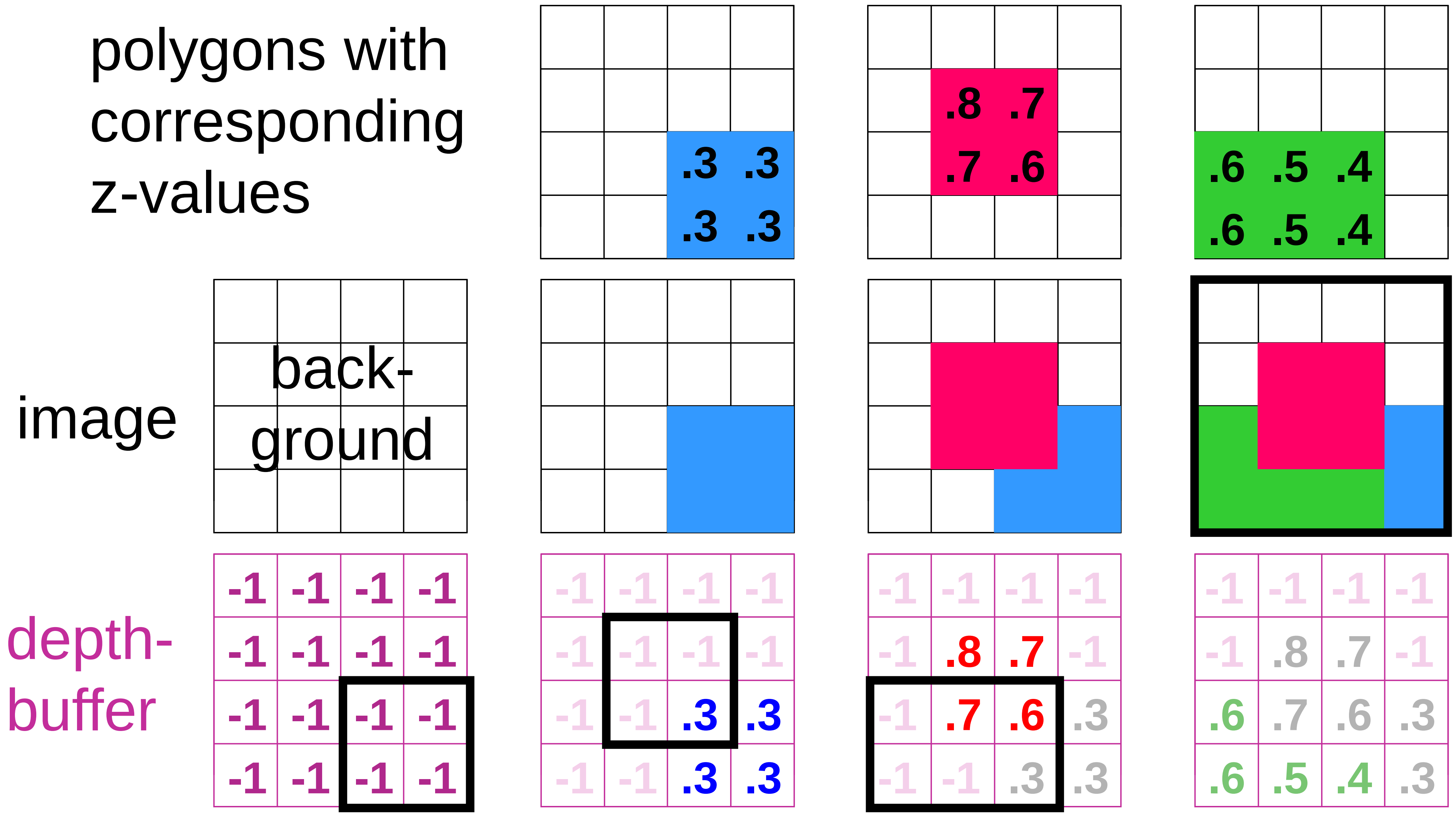
- refresh buffer (intensity information)
- depth buffer (distance information)

size corresponds to screen resolution
(for every pixel: r, g, b, z)

draw something =

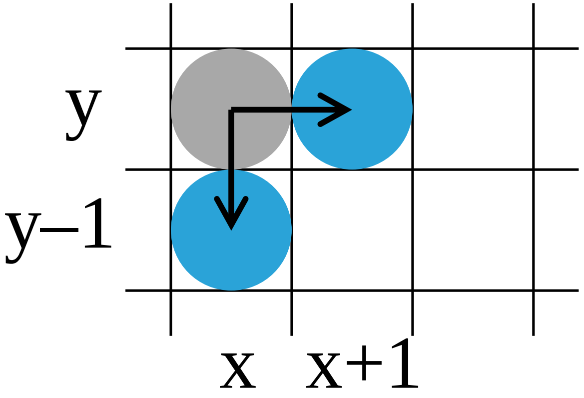
- **compare z with z in buffer**
- **if z closer to viewer**
- **then draw and update z in buffer**
- **else nothing!**

Depth-Buffer Algorithm Example



Depth-Buffer Algorithm

```
for all (x,y)
   frameBuff(x,y) = backgroundColor
   depthBuff(x,y) = -1
for each polygon P
   for each position (x,y) on polygon P
      calculate depth z
      if z > depthBuff(x,y) then
       depthBuff(x,y) = z
       frameBuff(x,y) = surfColor(x,y)
```



$$Ax + By + Cz + D = 0$$

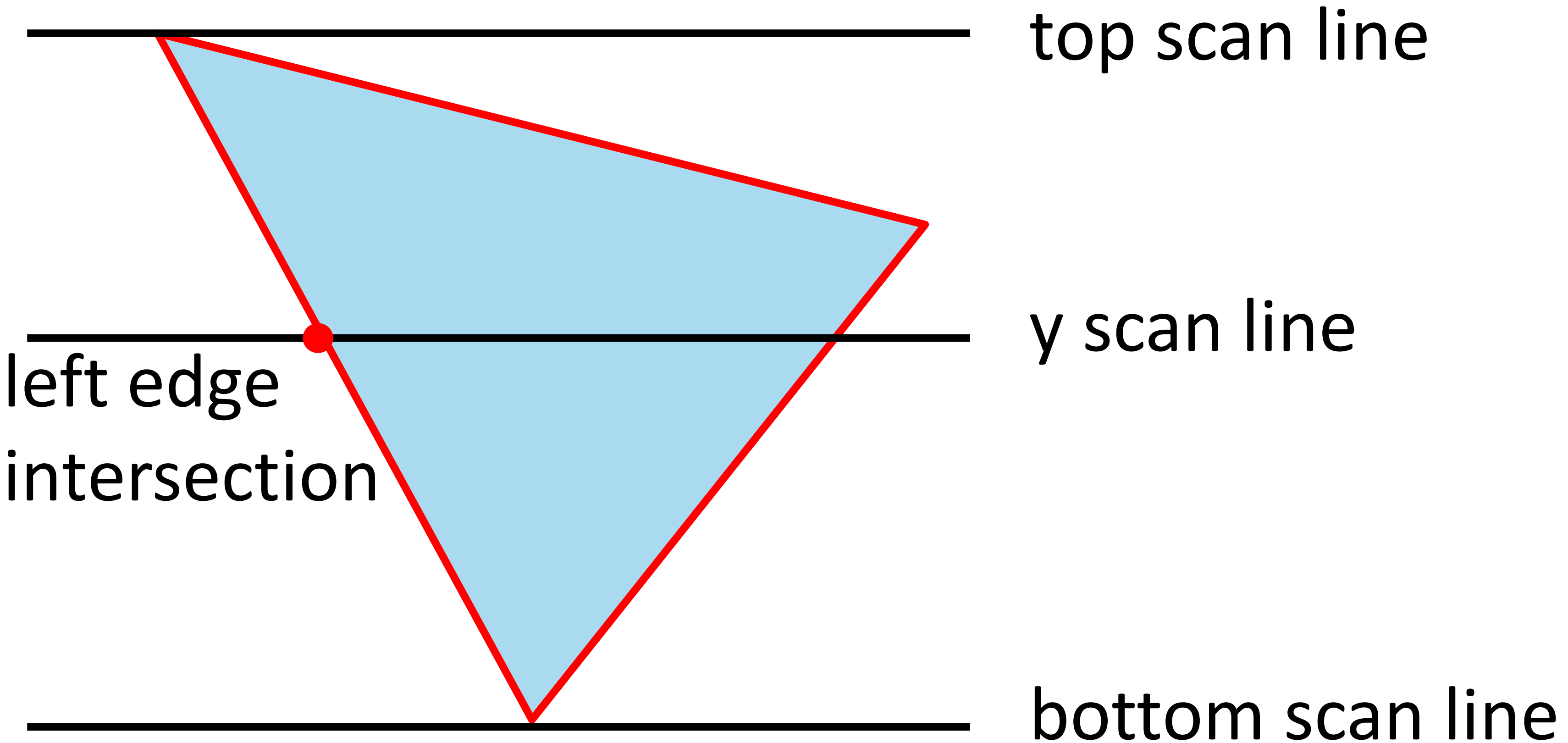
depth at (x,y): $z = \dfrac{-Ax-By-D}{C}$ **constants !**

depth at (x+1,y): $z' = \dfrac{-A(x+1)-By-D}{C} = z - \dfrac{A}{C}$

depth at (x,y-1): $z'' = \dfrac{-Ax-B(y-1)-D}{C} = z + \dfrac{B}{C}$

Depth-Buffer: y-Coordinate Intervals

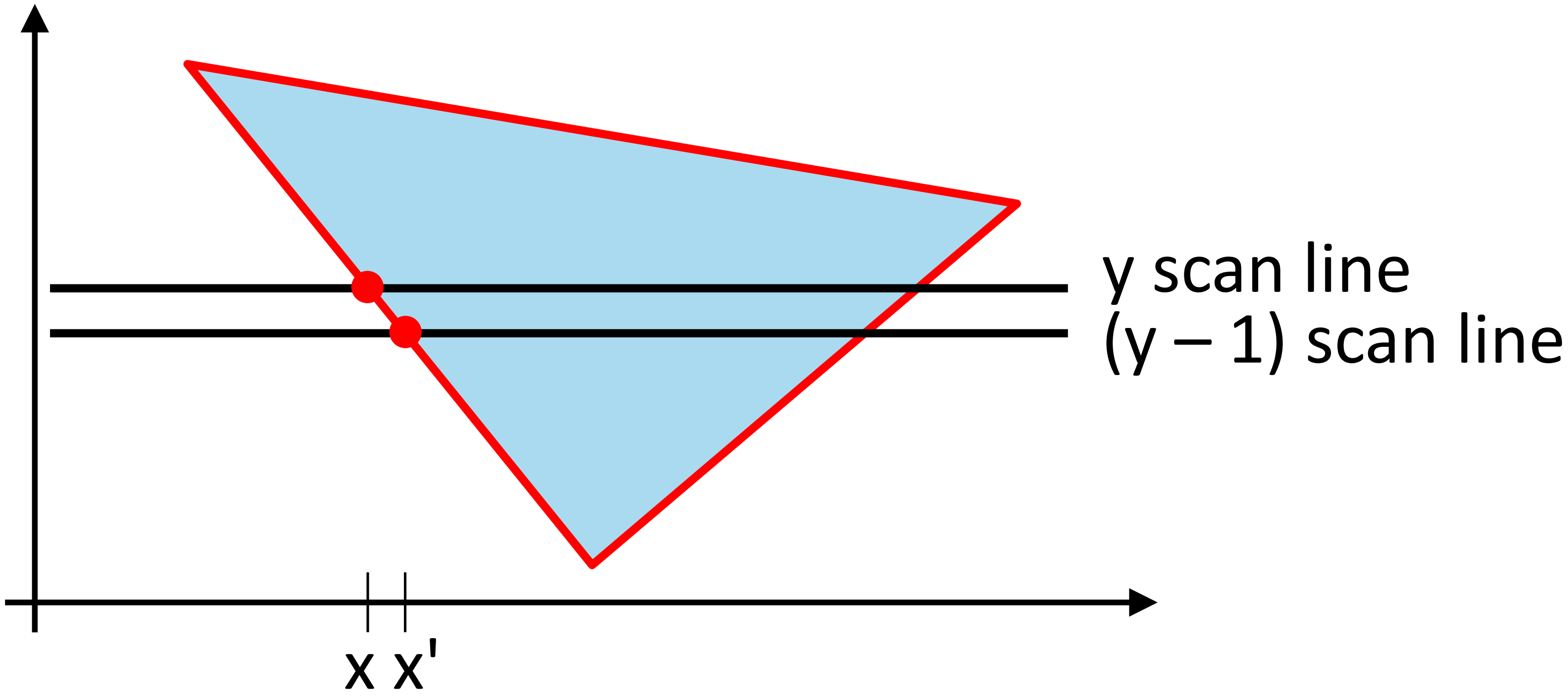
determine y-coordinate extents of polygon P



Depth-Buffer: Values down an Edge

$$z = \frac{-Ax-By-D}{C}$$

$$y' = y - 1$$
$$x' = x - 1/m \quad \Rightarrow \quad z' = \frac{-A(x-1/m)-B(y-1)-D}{C}$$

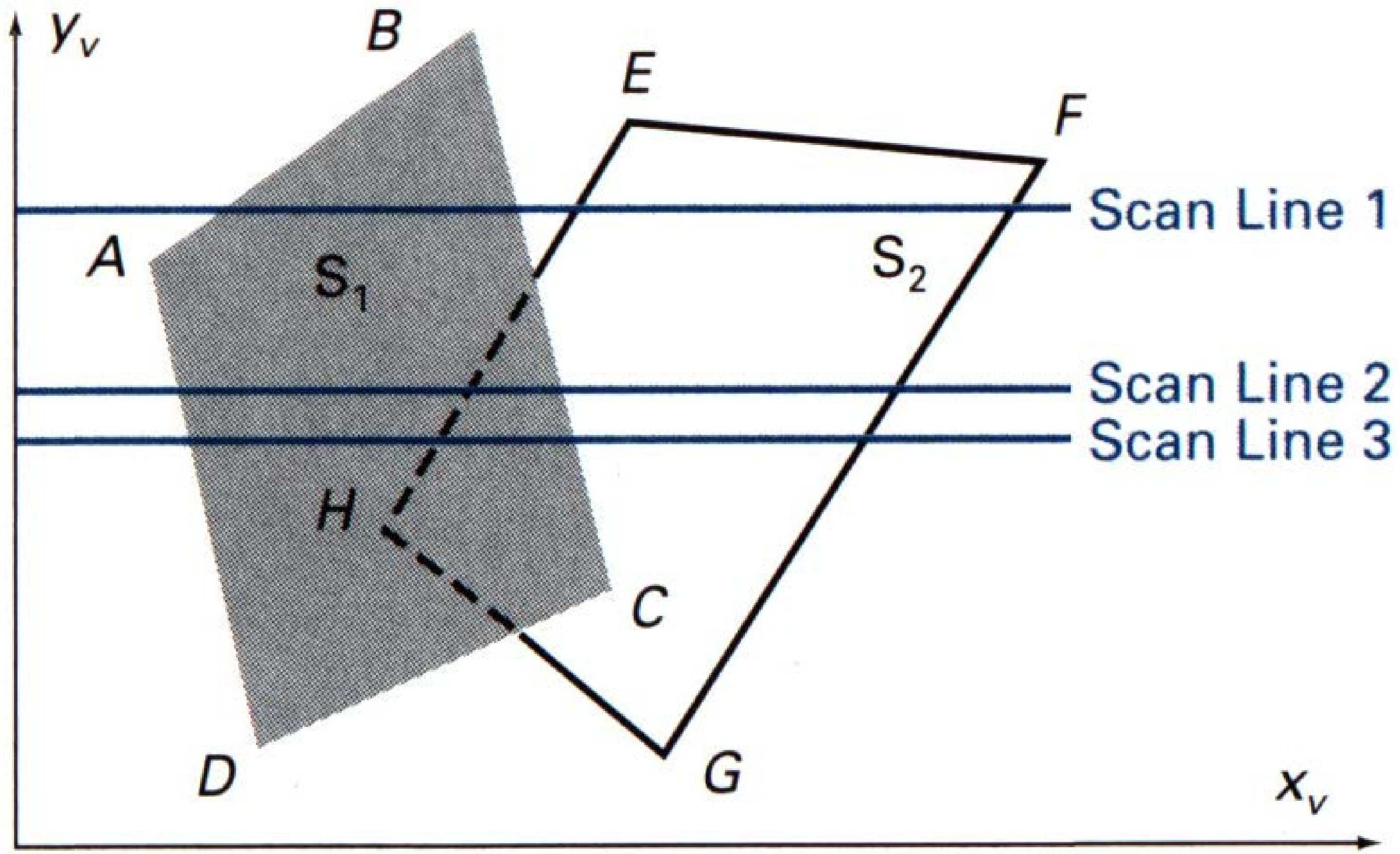


$$= z + \frac{A/m + B}{C}$$

constant !

Scan-Line Method

- image-space method
- extension of scan-line algorithm for polygon filling



Scan-Line Method: Edge & Polygon Tables

edge table (all edges, y-sorted)

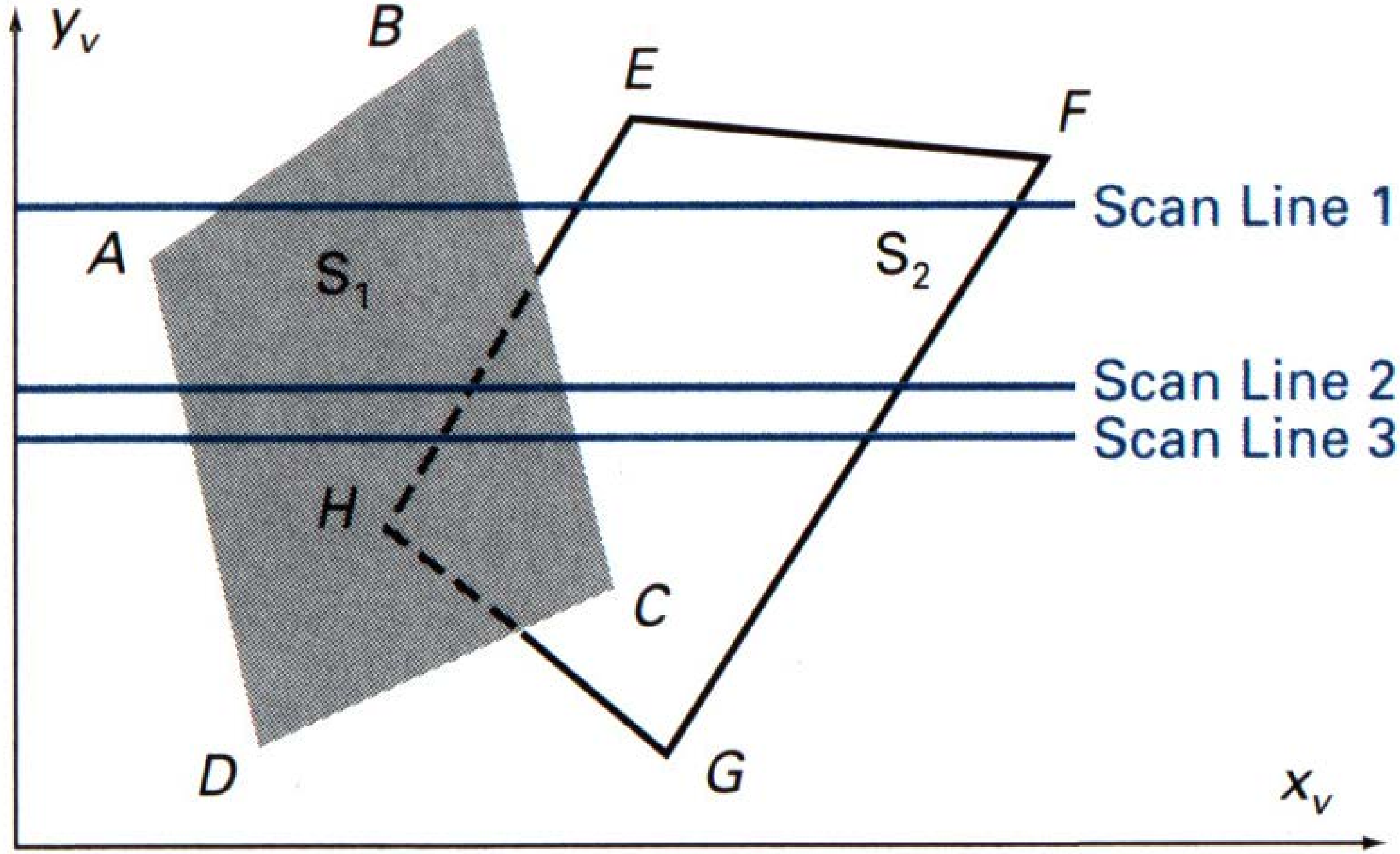
- coordinate endpoints
- inverse slope
- pointers into polygon table

polygon table (all polygons)

- coefficients of plane equation
- intensity information
- (pointers into edge table)

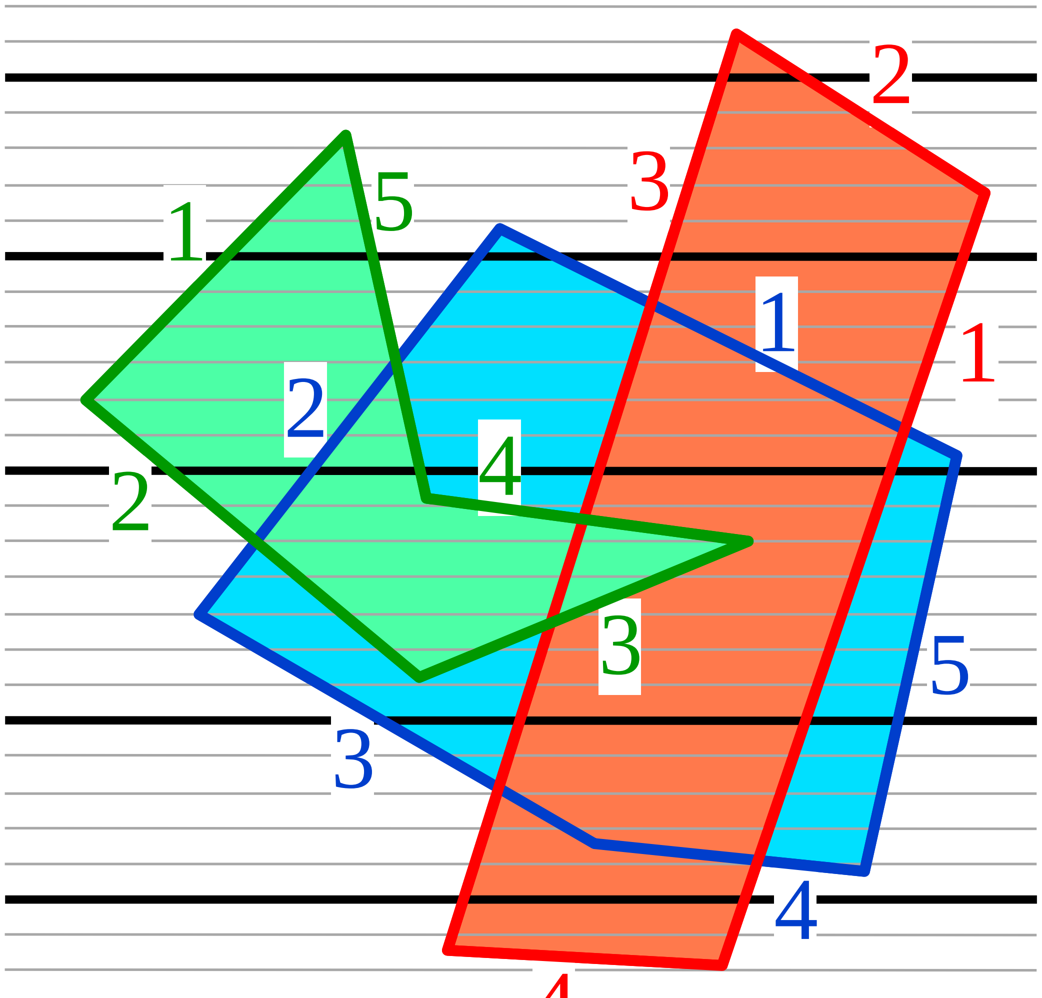
Scan-Line Method: Active Edge List

active edge list (all edges crossing current scanline, x-sorted, flag)



Scan-Line Method Example

Edge Table: 2,3,1,5,1,1,2,2,5,4,3,3,4,4 **Polygon Table: ■, ■, ■**



active edges	active polygons
3,2	■
1,5,2,1,3,1	■, ■, ■
2,2,5,3,1,5	■, ■, ■
3,3,1,5	■, ■
3,1	■

Scan-Line Method - Coherence

coherence between adjacent scan lines

- incremental calculations
- active edge lists very similar
  (easy sorting, avoid depth calculations)

Depth-Sorting Method: Overview

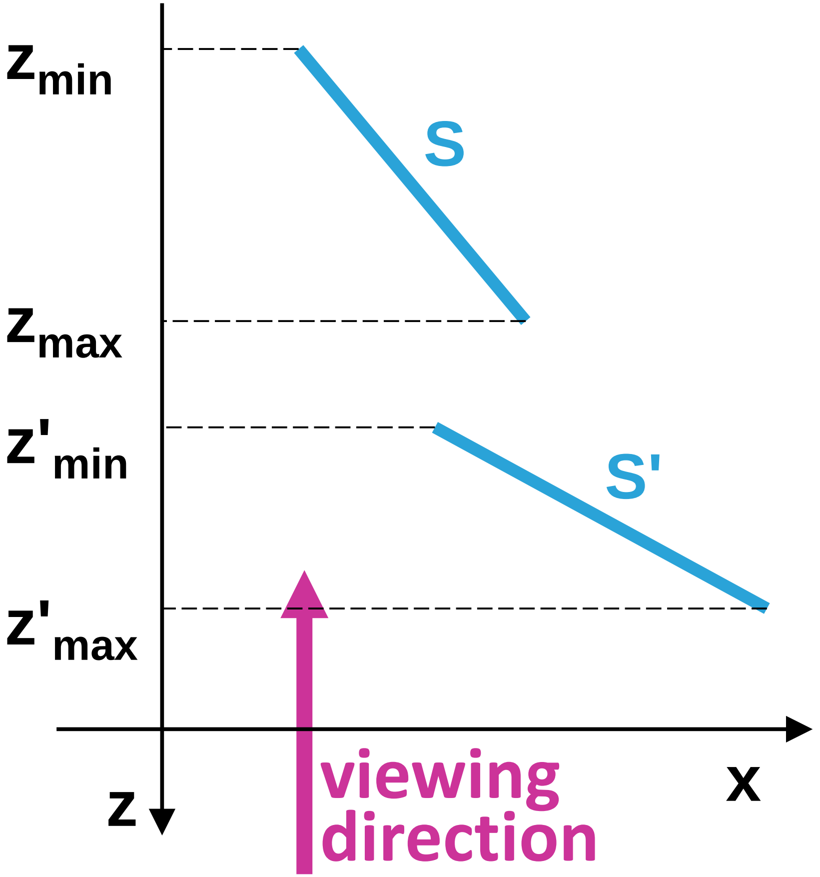
- surfaces sorted in order of decreasing depth (viewing in $-z$-direction)
  - "approximate"-sorting using smallest z-value (greatest depth)
  - fine-tuning to get correct depth order
- surfaces scan converted in order
- sorting both in image and object space
- scan conversion in image space
- also called "painter's algorithm"

Depth-Sorting Method: Sorting (1)

surface S with greatest depth is compared to all other surfaces S'

- no depth overlap → ordering correct
- depth overlap → do further tests in increasing order of complexity

2 surfaces with no depth overlap

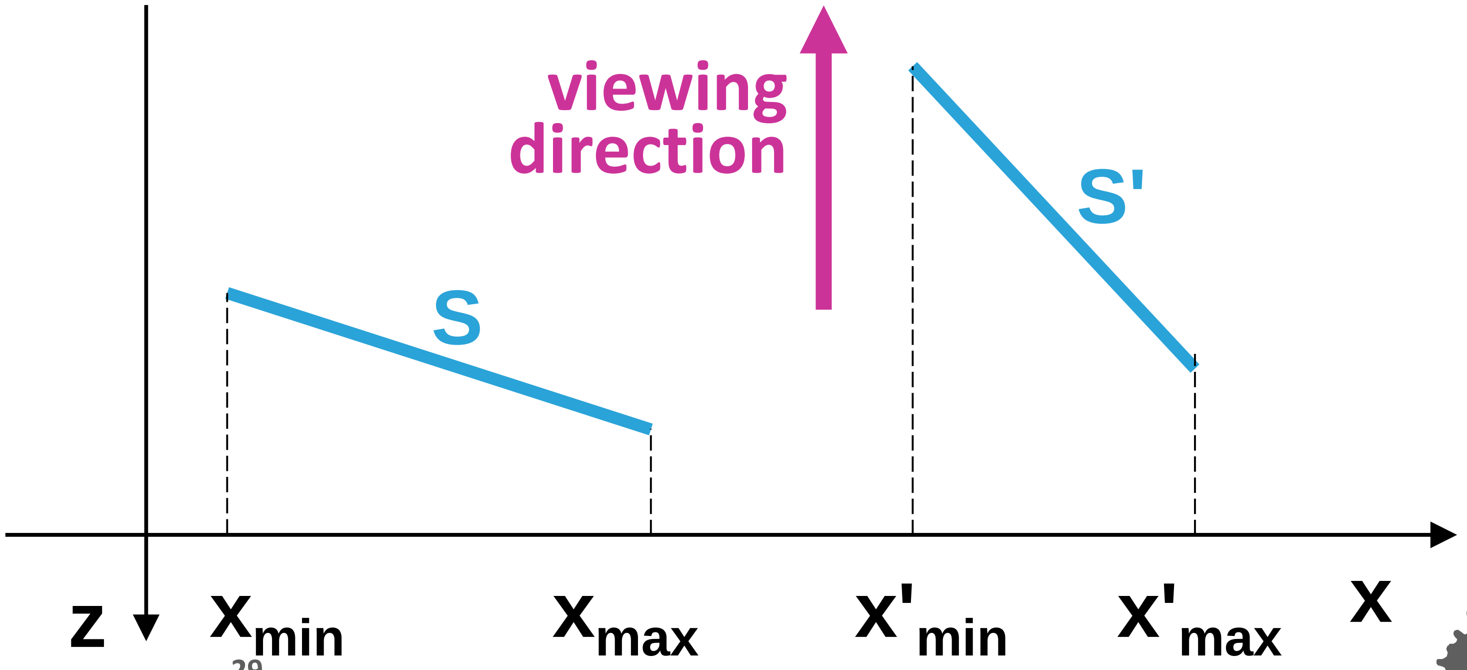


Depth-Sorting Method: Sorting (2)

ordering is correct if bounding rectangles in xy-plane do not overlap

$\rightarrow$ check x-,y-direction separately

2 surfaces with depth overlap but no overlap in the x-direction

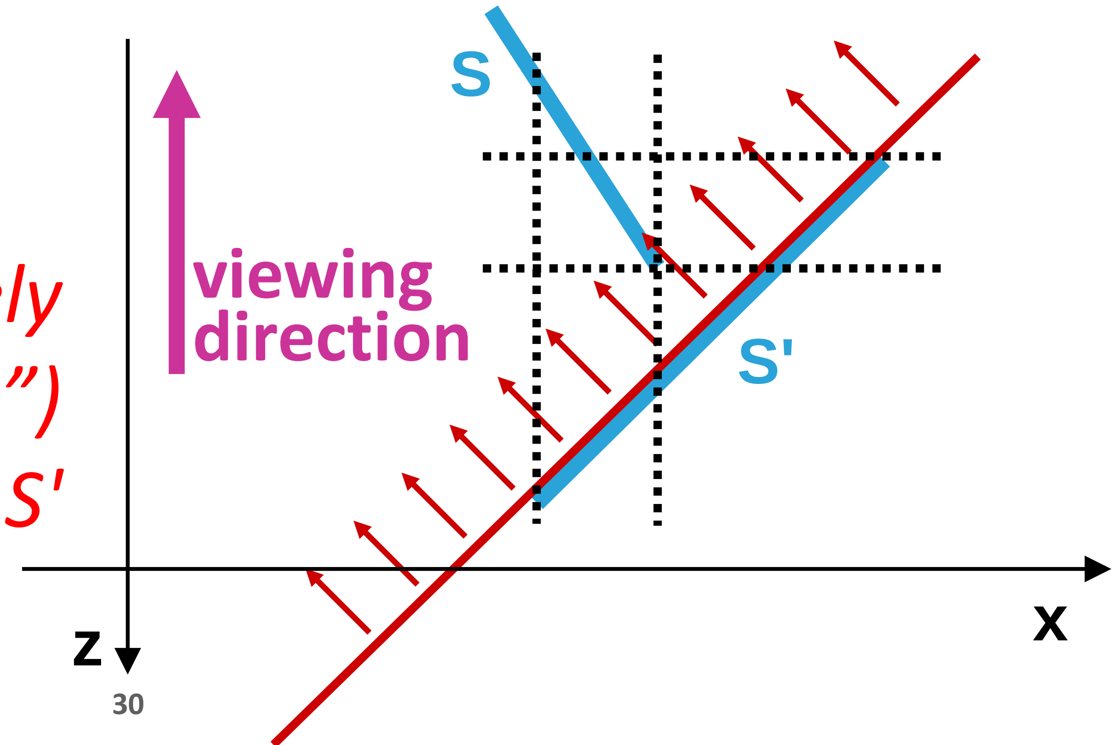


Depth-Sorting Method: Sorting (3)

ordering is correct if S completely behind S'

→ substitute vertices of S into equation of S'

S is completely behind ("inside") the overlapping S'

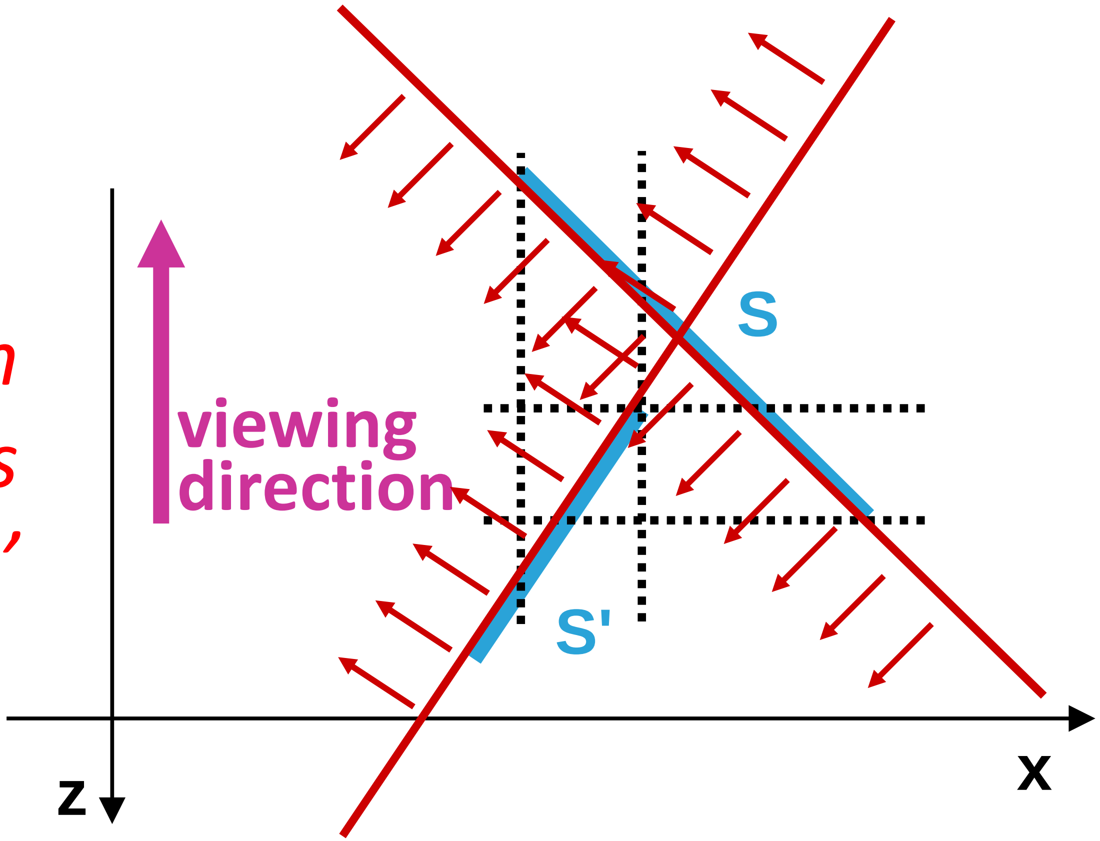


Depth-Sorting Method: Sorting (4)

ordering is correct if S' completely in front of S

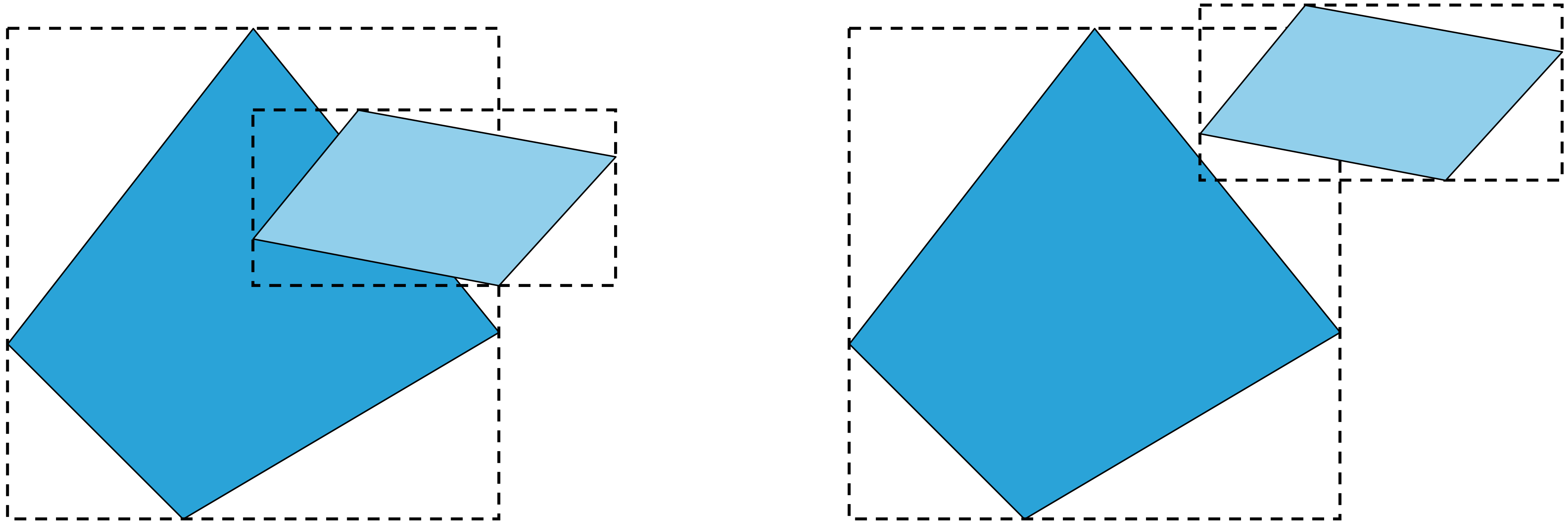
$\rightarrow$ substitute vertices of S' into equation of S

overlapping S' is completely in front ("outside") of S, but S is not completely behind S'



Depth-Sorting Method: Sorting (5)

ordering is correct if projections of S, S' in xy-plane don't overlap



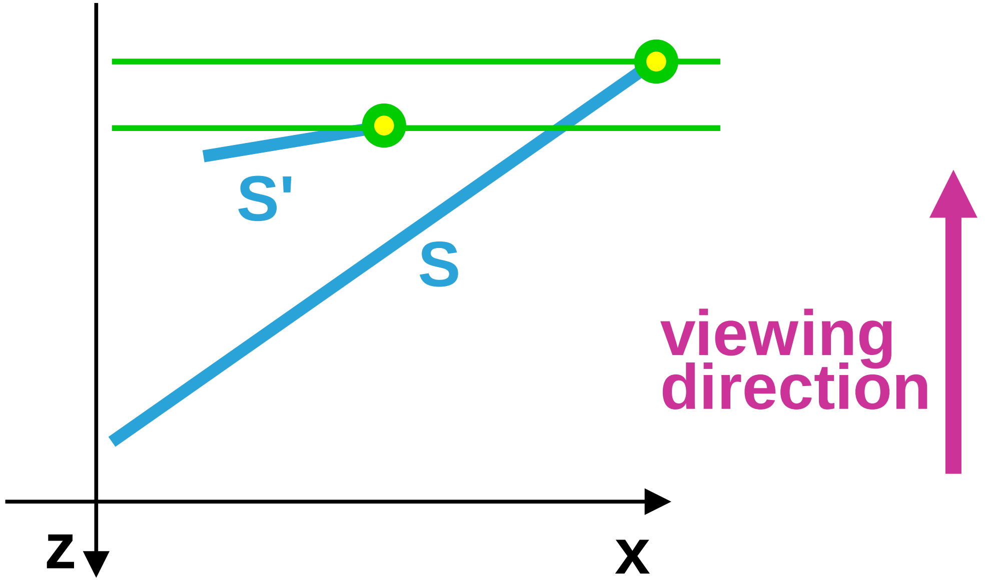
surfaces with overlapping bounding rectangles

Depth-Sorting Method: Sorting (6)

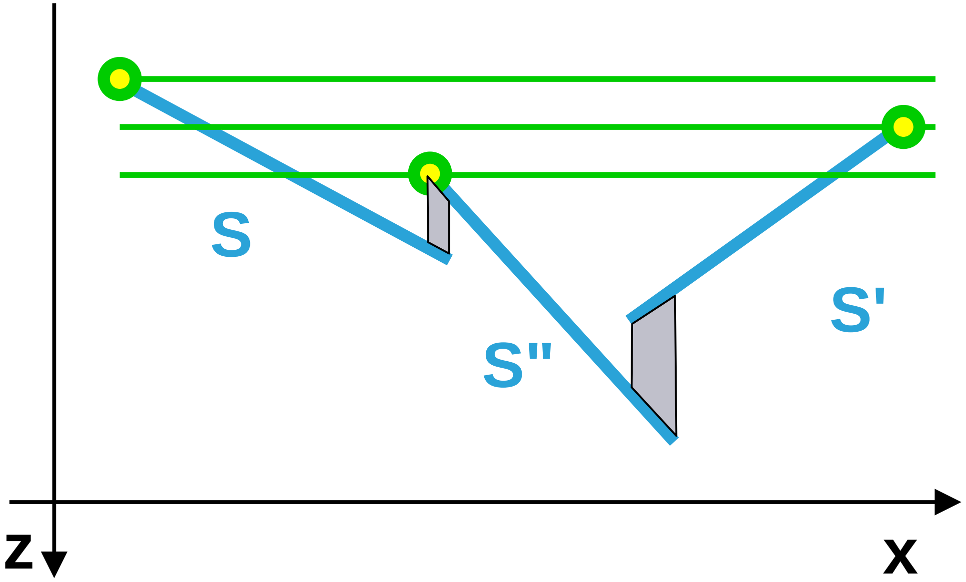
all five tests fail $\Rightarrow$ ordering probably wrong

$\rightarrow$ interchange surfaces S, S'

$\rightarrow$ repeat process for reordered surfaces



surface S has greater depth but obscures S'



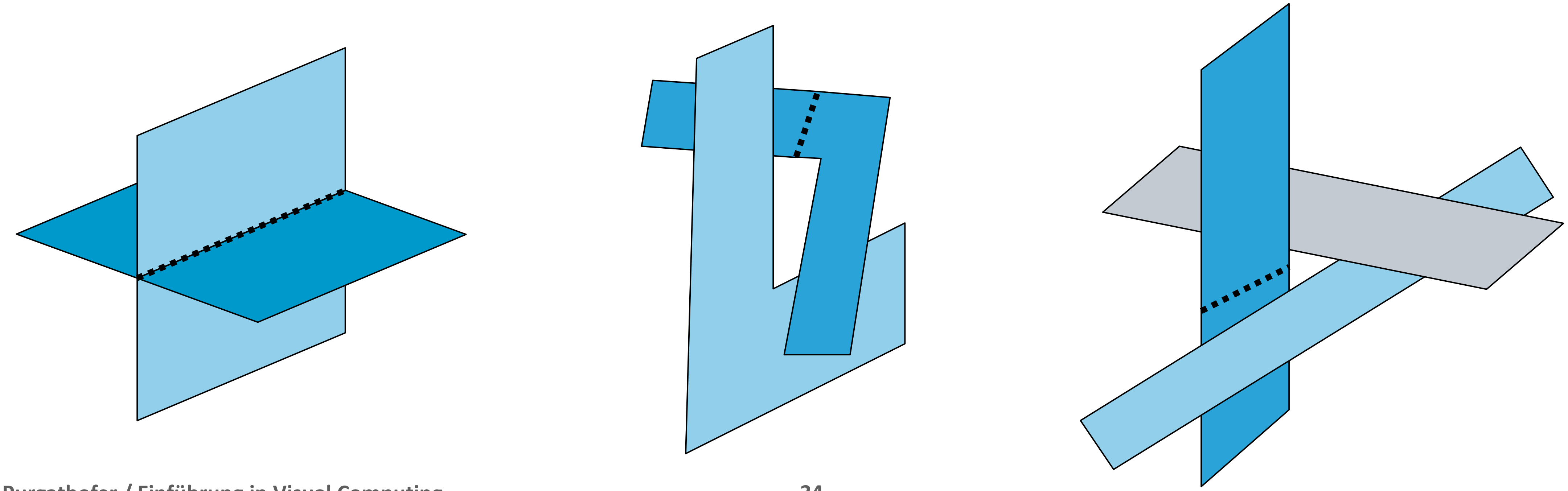
sorted surface list: S, S', S'' should be reordered: S', S'', S

Depth-Sorting: Special Cases

avoiding infinite loops due to cyclic overlap

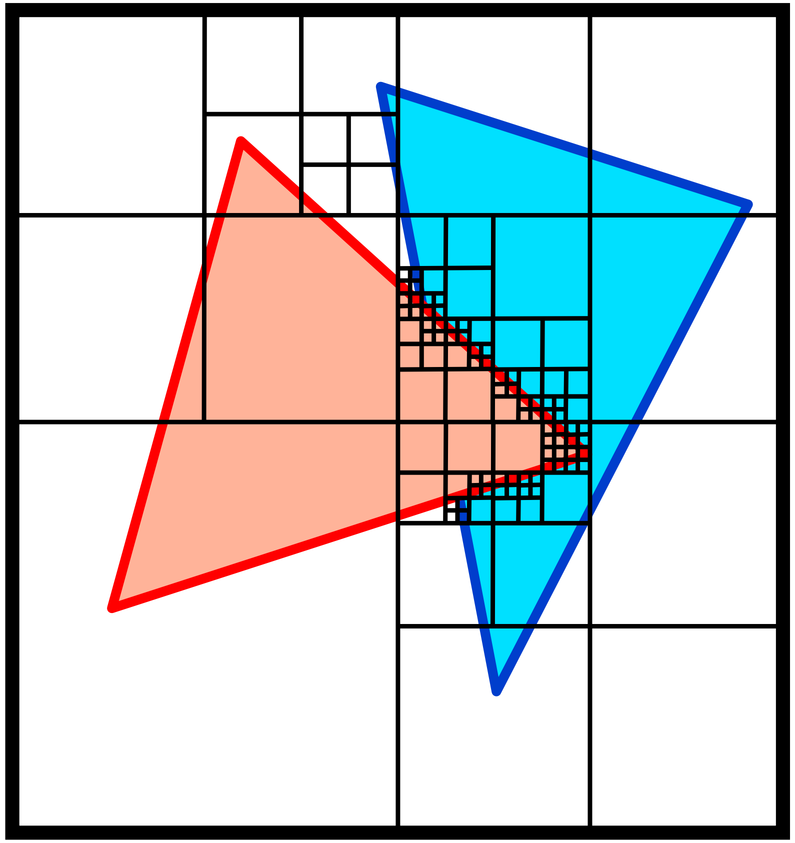
- reordered surfaces S' are flagged
- if S' would have to be reordered again $\Rightarrow$ divide S' into two parts

intersecting or cyclically overlapping surfaces!



Area-Subdivision Method (1)

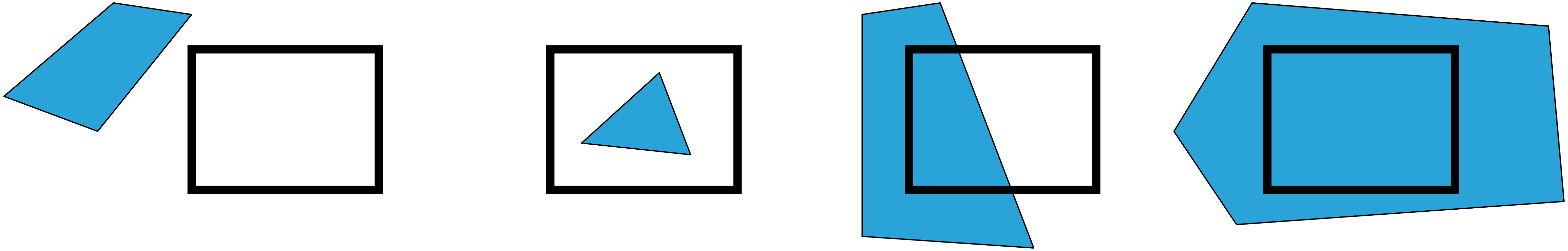
- image-space method
- area coherence exploited
- viewing area subdivided until visibility decision very easy



Area-Subdivision Method (2)

relationship polygon ⇔ rectangular view area

outside surface | inside surface | overlapping surface | surrounding surface

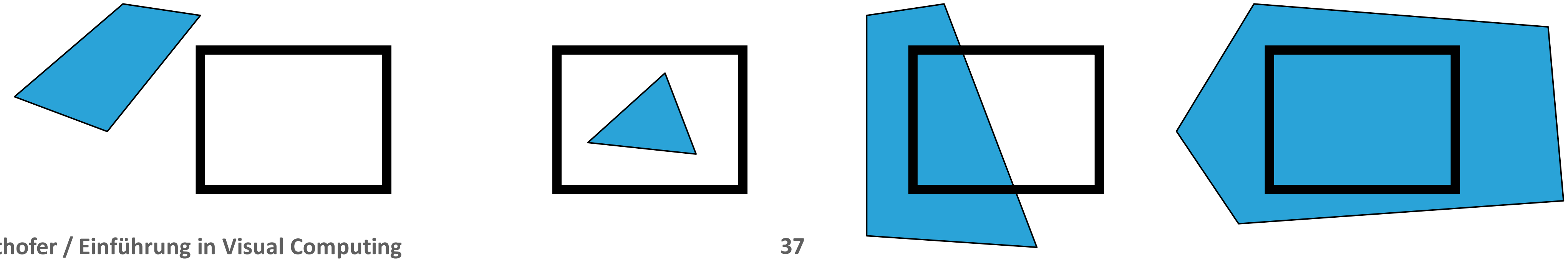


→ only these four possibilities

Area-Subdivision Method (3)

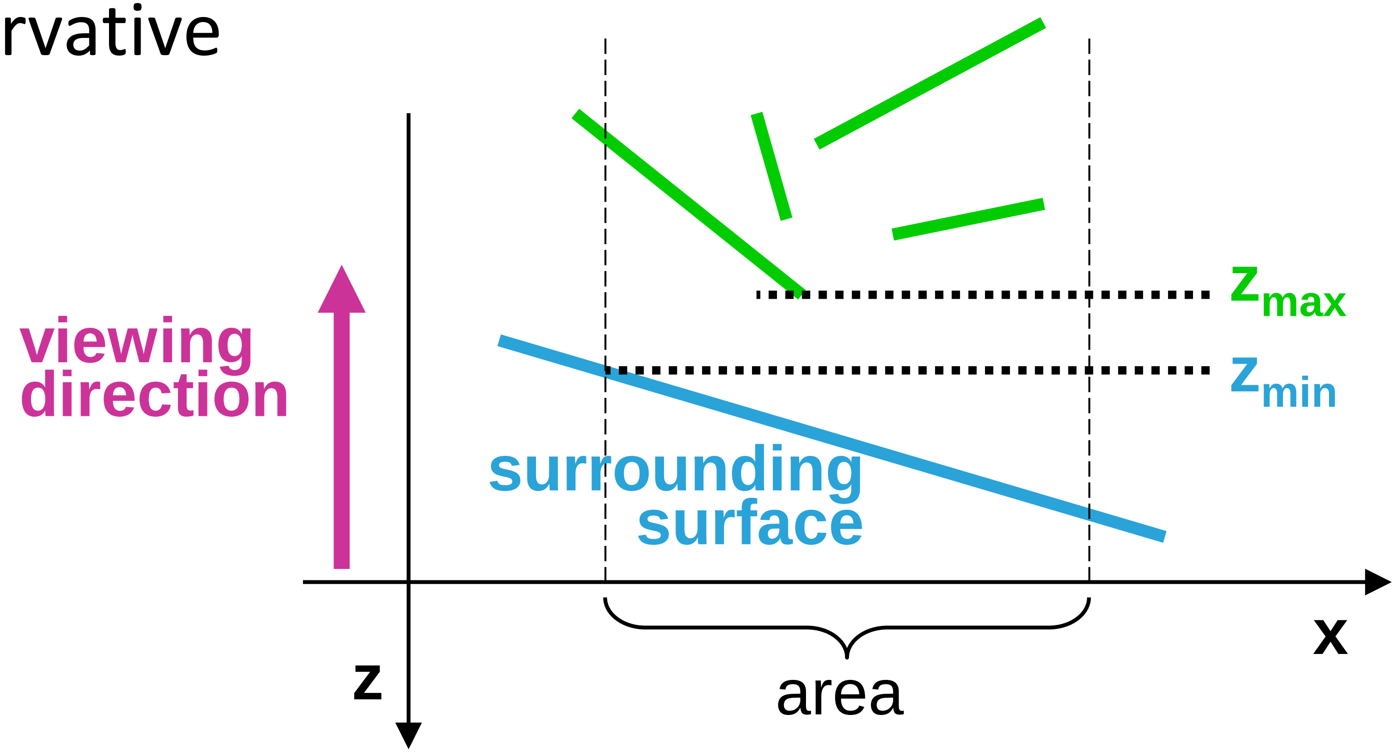
three easy visibility decisions:

- *all* surfaces are *outside* of viewing area
- only *one inside*, overlapping, or surrounding surface is in the area
- one *surrounding* surface obscures all other surfaces within the viewing area



Area-Subdivision Method (4)

- a surrounding obscuring surface
  - surfaces ordered according to minimum depth
  - maximum depth of surrounding surface closest to view plane?
  - test is conservative



Area-Subdivision Method (5)

if all three tests fail $\Rightarrow$ do *subdivision*

$\rightarrow$ subdivide area into four equal subareas

$\rightarrow$ outside & surrounding surfaces will keep status for all subareas

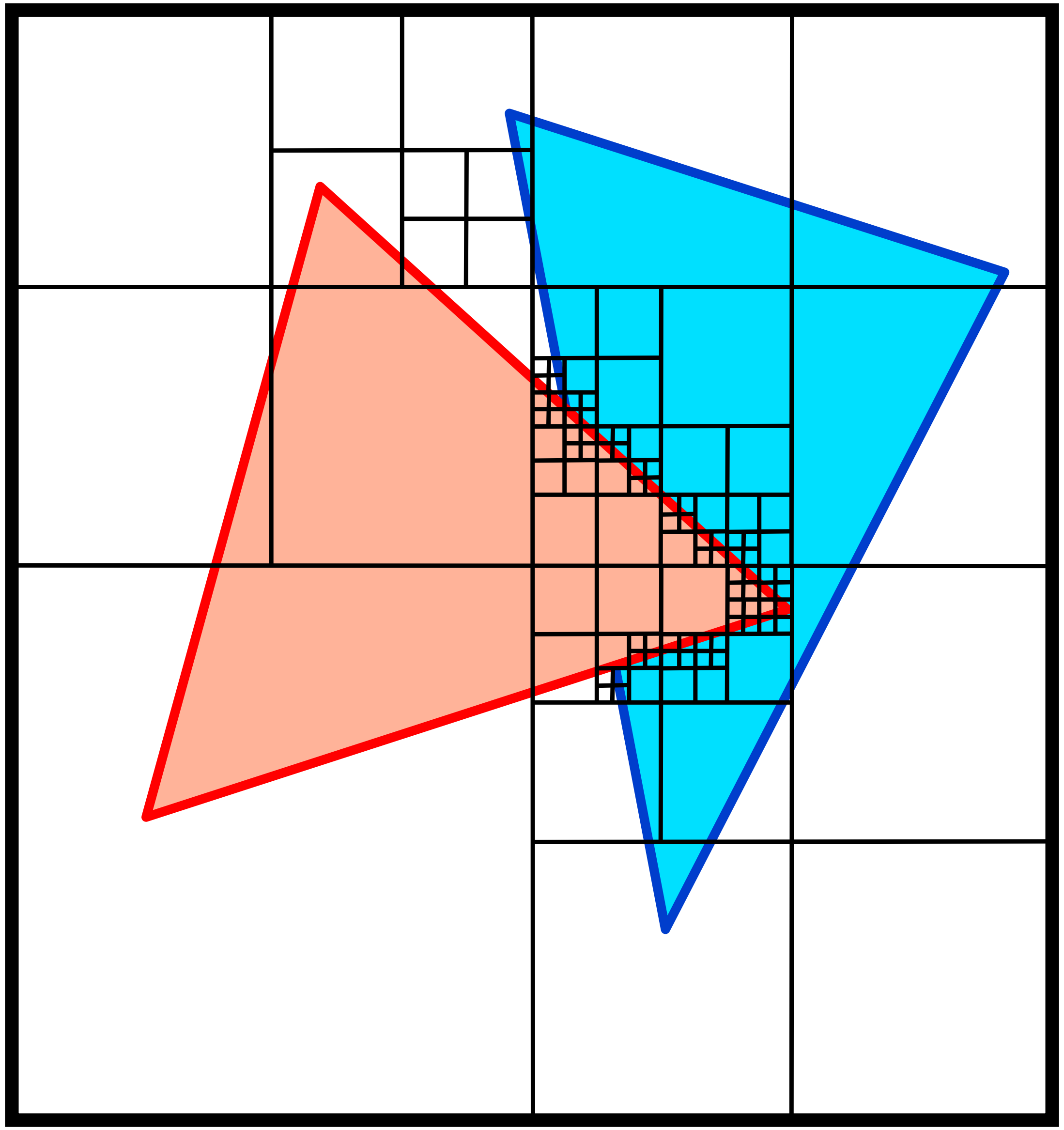
$\rightarrow$ some inside and overlapping surfaces will be eliminated

if no further subdivision possible (pixel resolution reached)

$\rightarrow$ sort surfaces and take intensity of nearest surface

Area-Subdivision Method Example

1 2
3 4



Octree Methods

recursive traversal of octree

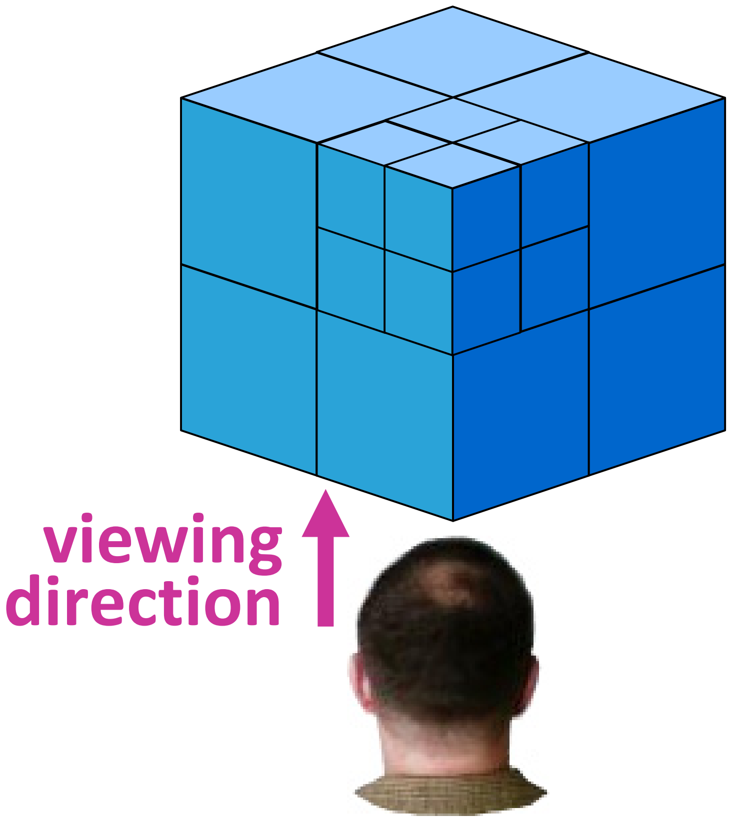
- traversal order depends on processing direction

front-to-back:

- pixel(x,y) written once
- completely obscured nodes are not traversed

back-to-front:

- painter's algorithm



Octree Methods

recursive traversal of octree

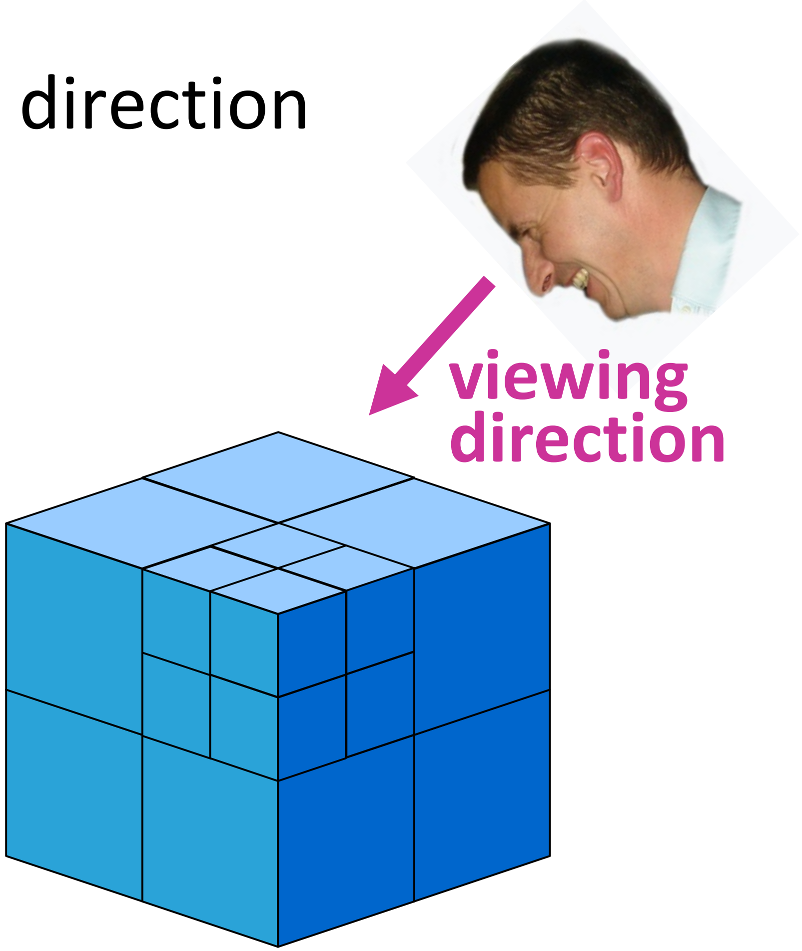
- traversal order depends on processing direction

front-to-back:

- pixel(x,y) written once
- completely obscured nodes are not traversed

back-to-front:

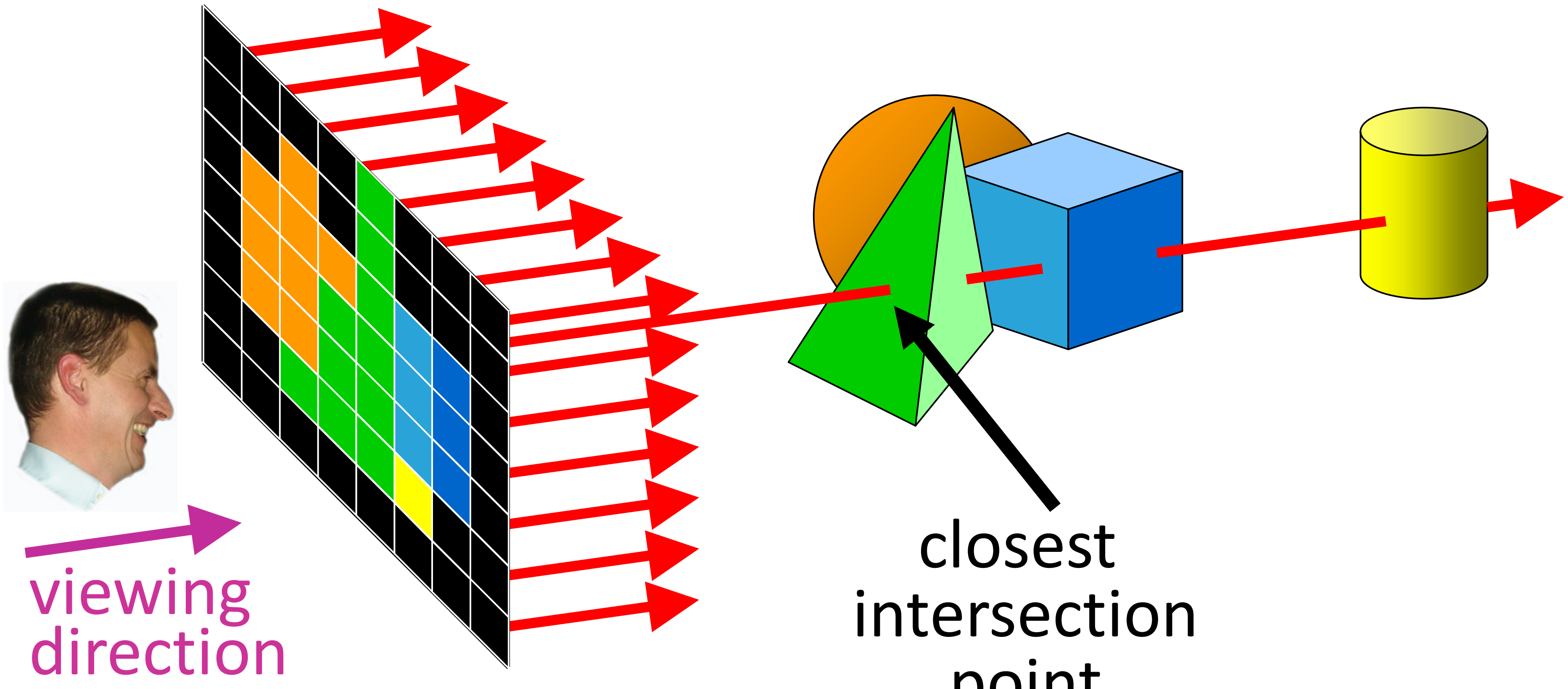
- painter's algorithm



Ray-Casting Method (1)

line-of-sight of each pixel is intersected with all surfaces

→ take closest intersected surface



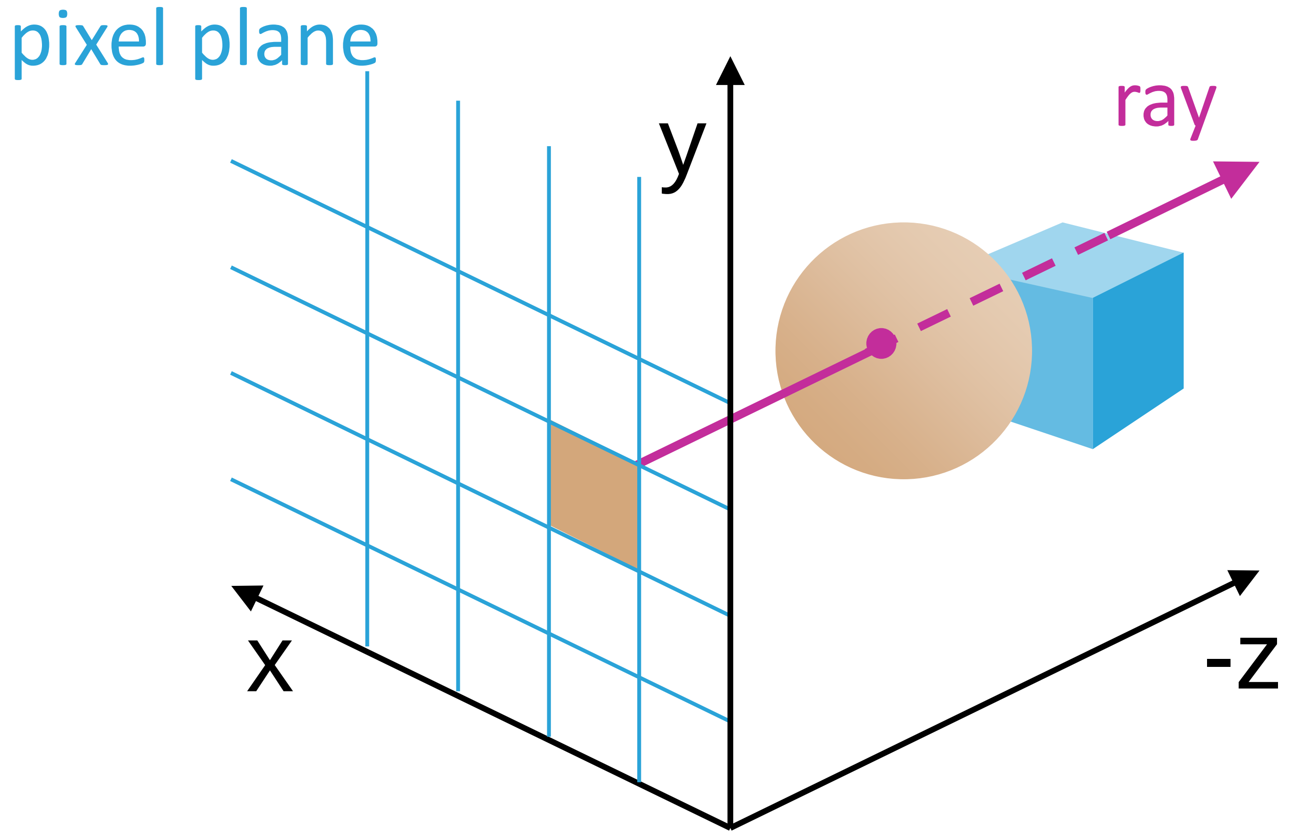
Ray-Casting Method (2)

- based on geometric optics, tracing paths of light rays
- backward tracing of light rays
- suitable for complex, curved surfaces
- special case of ray-tracing algorithms
- efficient ray-surface intersection techniques necessary

  → intersection point & normal vector needed

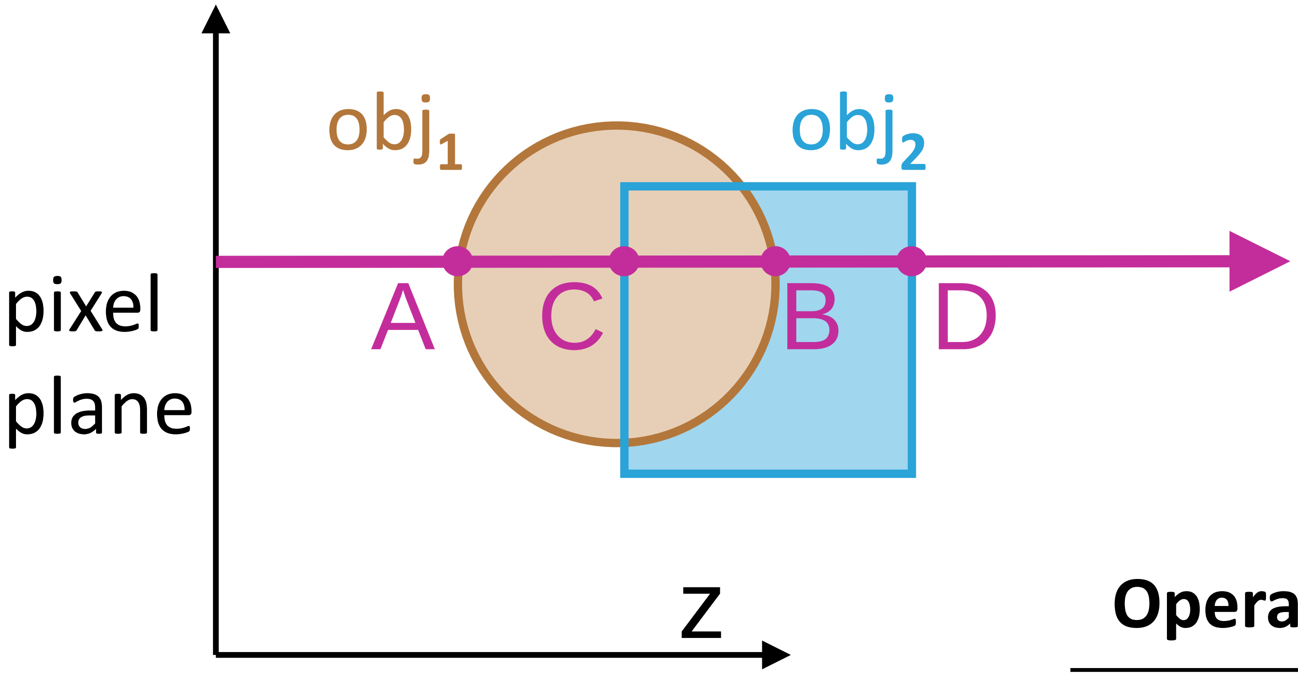
Ray-Casting Methods for CSG (1)

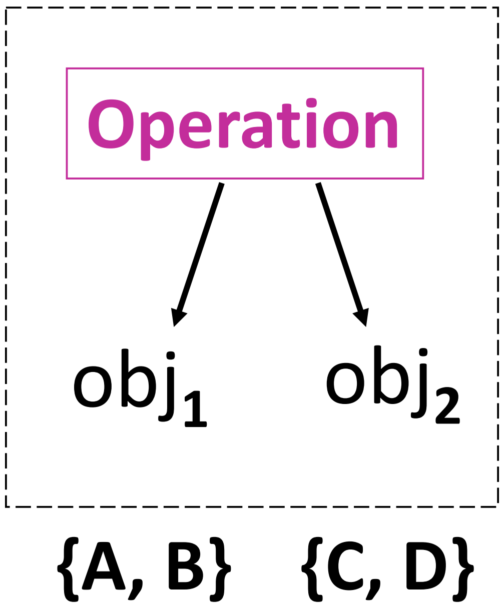
visibility processing



Ray-Casting Methods for CSG (2)

determining surface limits





Operation	Result
Union	{A, D}
Intersection	{C, B}
Difference	{A, C}

volume determination

$$V_{ij} \approx A_{ij} \cdot \Delta z_{ij} \qquad V \approx \sum V_{ij}$$